TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ANISHA SAMPATH KUMAR

AN INTEGRATED APPROACH FOR ONTOLOGY-DRIVEN CONFIGURATION MANAGEMENT AND RUN-TIME EXECUTION OF MANUFACTURING SYSTEMS

Master of Science thesis

Examiner: prof. Jose L. Martinez Lastra
Examiner and topic approved by the Faculty Council of the Faculty of Engineering Sciences
on 6th May 2015

ABSTRACT

ANISHA SAMPATH KUMAR: An Integrated Approach for Ontology-Driven Configuration Management and Run-Time Execution of Manufacturing Systems
Tampere University of technology
Master of Science Thesis, 83 pages
January 2016
Master's Degree Program in Machine Automation
Major: Factory Automation
Examiner: Professor Dr. Jose L. Martinez Lastra
Supervisor: Dr. Andrei Lobov

Keywords: Semantics, ontology, query, service-oriented architecture, factory automation, OWL, SPARQL

The contemporary manufacturing systems must respond instantly to rapidly changing customer and market requirements in order to survive the intensive competitive environment. The factories should have the agility to adapt to mass customization and introduction of new product, equipment and technology to manufacturing. This is possible by having re-configurable loosely coupled system which offers run-time decision making capability on all levels of factory from shop floor to ERP systems. This thesis proposes a knowledge-based approach for achieving shop-floor device configuration management, run-time execution support for orchestration engines and re-configurable visualization for monitoring systems.

The thesis work was carried out as a part of the European project eScop by Arthemis Joint Undertaking, where knowledge bases as the information source for manufacturing execution system is the core concept. This thesis work involves semantic modeling of the manufacturing knowledge in a Manufacturing System Ontology (MSO) and exposing the knowledge to other components of the system using web services. It employs Service Oriented Architecture (SOA) on device level to facilitate knowledge extraction. A methodology is put forward in the thesis to design ontology with broader capabilities and queries for reasoning the ontology. Ontologies are extendable and easy to update offering flexibility to address system changes. The reusability of knowledge simplifies the addition of new product or equipment and thereby offering re-configurability to the system. The proposed approach has been tested by implementing on a real manufacturing system and the research objectives were achieved.

PREFACE

This thesis work is the result of several months of tremendous effort, time and patience. During my work on the thesis, many people supported and encouraged me. I would like to express my gratitude for them in this preface.

I am grateful to prof. Jose L. Martinez Lastra for the support and opportunity to work in an interesting project. As well I am thankful to all the members of FAST Lab who guided and encouraged me to complete the thesis work.

My very special thanks to the thesis supervisor Dr. Andrei Lobov for his endless patience, advice and encouragement. I would also like to express my gratitude for the whole eScop team and especially Sergii and Wael for their great cooperation and support for the thesis.

Also I would like to thank my dear friends Veerendra, Ashwini, Fernanda and Maya for their continuous support and motivation.

Finally, I would like to express the most kind and special gratitude to my parents and brother for their guidance and support in every aspect.

Tampere, 22.1.2016

Anisha Sampath Kumar

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AGV	Automatic Guided Vehicle
AI	Artificial Intelligence
BPMN	Business Process Modeling Notation
CPS	Cyber-Physical Systems
CSS3	Cascading Style Sheets 3
DPWS	Device Profile for Web Services
ERP	Enterprise Resource Planning
EU	European Union
FBS	Function-Behavior-Structure
FIPA	Foundation for Intelligent Physical Agents
FIS	Factory Information System
GDP	Gross Domestic Product
HMI	Human Machine Interface
HTML	Hyper Text Markup Language
ICT	Information and Communication Technology
IT	Information Technology
JSON	JavaScript Object Notation
KIF	Knowledge Information Format
MES	Manufacturing Execution System
MIRA	Modular, Intelligent and Real-time Agent
MSO	Manufacturing System Ontology
ORL	Orchestration layer
OWL	Web Ontology Language
PHL	Physical Layer
RDF	Resource Description Framework
RPL	Representation layer
SCADA	Supervisory Control And Data Acquisition
SHOE	Simple HTML Ontology Extensions
SOA	Service-Oriented Architecture
SOCRADES	Service-Oriented Cross-layer Architecture for Distributed smart Embedded Devices
SPARQL	SPARQL Protocol and RDF Query Language
SVG	Scalable Vector Graphics
SWRL	Semantic Web Rule Language
UI	User Interface
UP	Unified Process
UPON	Unified Process for Ontologies
VIS	Visualization layer
W3C	World Wide Web Consortium
WS	Web Service
XML	Extendable Markup Language
XOL	eXtensible Out-of-band Language

1. INTRODUCTION

Manufacturing is the key driver of jobs and growth in Europe. At present manufacturing is challenged by increasing scarcity of resources, handling huge amount of data and mass customization. Advancements in manufacturing sector like sustainable manufacturing technologies and ICT-enabled intelligent manufacturing have resulted in smart factories and aims to improve the competitiveness of the EU's manufacturing industries. The global market for industrial automation solutions has increased from $155 billion in 2011 to $190 billion in 2015[1]. The European Commission has set its goal to increase the contribution European industry makes to EU GDP from the 15% in 2014 to 20% by 2020. For this purpose, the commission has been taking actions to stimulate investments in new technologies for manufacturing [2][3].

1.1 Motivation

In today's intensely competitive environment, manufacturing industries must anticipate and respond instantly to changing customer and market requirements. Many companies are currently experiencing increasing demands from their customers for the delivery of customized products with the same delivery time, price and quality as mass produced products. Mass customization leads to revision of the company's overall business model. The smart factories are able to quickly define and produce products based on market requirements by adopting an efficient configuration management process.

The manufacturing is highly dynamic in nature and hence the changes are both inevitable and frequent. Poor configuration management affects company's ability to manage changes effectively because they have great difficulty assessing the full impact of the proposed changes. They have inability to re-use existing configuration of the system to meet the desired changes. This is because the existing configuration of the system is not adequately documented. The speed of developing a product can be greatly enhanced if the previous product configuration was reused [4]. But, ineffective configuration management requires engineers to recreate the existing product components which wastes valuable resources that would be better targeted towards creating new products. Today's economic environment requires companies to have a configuration management system that can quickly re-configure the system simplifying the addition of new product or equipment.

By utilizing new technologies to optimize the configuration management, the industries can improve the efficiency of their product development process in terms of significant reduction in time-to-market, reduced product cost and increased product quality. The

new technologies offer more effective means to define and produce new products. The advancements in Information and Communication Technology (ICT) help industries to maximize the efficiency of manufacturing by reducing the paper work and optimizing manufacturing and product flow of industrial processes.

To support automatic re-configuration and run-time execution of manufacturing system, information regarding the system on various levels should be made available. As discussed in [5], the development in ICT has increased the IT support on all hierarchical levels of the factory, emerging a new trend today with increased information flows across all levels of the factory. By employing Service Oriented Architecture (SOA) paradigm, the information and functionality of the shop floor devices can be exposed to information systems. The smart devices are provided with the opportunity to offer self-descriptions by the device itself. Web Service (WS) technologies like RESTful services or WS-* protocols (SOAP Web Services) are widely employed to realize SOA [6]. Thus by adopting new technologies in manufacturing, the information on all levels of factory can be exposed as services.

1.2 Problem definition

To achieve effective configuration management and run-time support for manufacturing system, the necessary information about the system should be made available. Nowadays, the need for Factory Information System (FIS) has become inevitable. Rapid communication data and sufficient information is vital to make the right decision at the right time. The problem is not that there is not enough data but the data is not accessible. It is typically locked up in a variety of different systems ranging from computer applications to spreadsheets to paper records to machine themselves. The aim of FIS is to provide a smooth information flow within the manufacturing system and also considerably saves time and paper work compared to traditional information models [7].

Any change in the manufacturing system requires re-configuration of the system and this requires re-implementation of the information system in order to keep FIS synchronized with the manufacturing system. Re-implementing FIS requires a lot of time and cost. Moreover, the benefits of implementing modern technologies in manufacturing are limited by the time and cost for re-configuration. Due to this, the need to automate the configuration process is inevitable.

The software level re-configuration of the system requires human engineering knowledge. This drives the need to have a knowledge model storing all the information about manufacturing system necessary for supporting automatic re-configuration.

1.2.1 Justification for the work

As discussed earlier, adopting new technologies benefits manufacturing but their implementation is limited due to the cost and time for re-configuration. The FIS has to be re-implemented to keep it synchronized with the manufacturing system. To save time and cost the manufacturing system should be automatically re-configured in the event of introduction of new product, equipment or technology. In order achieve this; semantic enrichment and reasoning can be used.

By having a knowledge representation for manufacturing system using ontology, all the manufacturing information can be stored in one place, with a standard universal format and re-used to automatically define a new product or equipment on their addition. The mechanism of querying the ontology helps to provide new configuration for the dynamically reconfigurable system. Due to this, the information system will be always synchronized with the manufacturing system. Moreover, it benefits the manufacturing as new technologies can be adopted easily without concern over the cost for reconfiguration. The advantages like re-usability, re-configurability and flexibility in manufacturing are offered by ontology models and it drives the need for them in manufacturing.

The existing knowledge models in manufacturing domain are small and very specific in their implementation focusing only on a particular concept of manufacturing. Hence there is a need for an integrated approach to develop generic ontology model for manufacturing. As discussed earlier, nowadays with the help of SOA paradigm and technologies like DPWS and RESTful Web Services, all the necessary information about the devices can be exposed and stored in ontology. The same concepts form the basis of the architecture which is implemented in Embedded Systems Service-based Control for open Manufacturing and Process Automation (eScop), EU project [8]. The thesis work proposes the development of manufacturing system ontology which together with queries supports the dynamic configuration management and execution of the system.

1.2.2 Problem statement

As discussed in the problem definition and the justification of work, ontology models are required for efficient configuration management and execution of the manufacturing system. There is a need for knowledge models to be used as a source of information so that the information can be re-used to support new configurations in dynamically reconfigurable systems. It also offers flexibility to the manufacturing system. Thus, an integrated approach is essential to realize configuration management and run-time execution support for the system. It should answer the following questions:

- How to represent the manufacturing knowledge in ontology? The manufacturing knowledge includes shop-floor device information, orchestration related information and visualization information.

- How to offer reasoning capability?
- How to update the information in ontology when there is a change in the system (during addition of new product, equipment, etc.)?
- How to reuse the information for creating new configurations at run-time?

1.3 Work description

1.3.1 Objectives

The objectives of this thesis work are provided below.

- *Defining the approach:* The approach forms the blueprint of this thesis work. It defines the steps for developing and using ontology and ontology services to achieve efficient configuration management and execution of system. Also defines the standards, tools and technologies needed to accomplish the task.
- *Designing an ontology model for manufacturing system:* This task involves development of an ontology design methodology and using it to represent the concepts of the manufacturing system in knowledge base. The developed manufacturing system ontology model should be generic and confront to standards of manufacturing system models.
- *Defining ways to update and retrieve information stored in MSO:* This task involves defining a standard approach for manipulating the information in MSO using queries.
- *Defining new configuration at run-time:* This task involves re-using the existing information about the product/equipment in ontology to create new configuration when new product/equipment is added.
- *Testing the developed approach on a use case:* This task involves experimental study on a real manufacturing system. It involves instantiation of the developed ontology for the particular use case system. Defining the run-time situations which require information from MSO for decision making. Defining standard templates for queries to manipulate the ontology. Defining the ontology services for other layers to insert/update the ontology and to expose the information to other layers for their execution.

1.3.2 Methodology

The research work will be carried out as follow. First, a theoretical background of the state-of- the-art concepts, technologies and tools related to thesis topic is presented. Then, the approach for developing ontology and ontology services is presented. A methodology is proposed for semantic modeling of the manufacturing knowledge in MSO and also defines the standards and tools for processing information stored in knowledge base. A generic ontology model with broader capability is developed.

Once the technology, model, tools and techniques are defined, the implementation is carried out using FASTory Production Line, an assembly system for mobile phones

available in Tampere University of Technology. The implementation involves testing the proposed approach to achieve the objectives of the thesis. Developed model is instantiation for the FASTory system. The required queries are created to manipulate the information in MSO. Then the ontology services are implemented which makes use of the queries to serve the request of other components i.e. updating knowledge in ontology or retrieving knowledge needed for their execution.

Finally, the results from implementation and research work in general are discussed and the research is concluded.

1.3.3 Assumptions and limitation

The assumptions and limitations made in this thesis work are

- The shop floor devices are offered with self-description capability. The smart devices follow a service oriented architecture using RESTful web services technology so that the information and functionality of the device can be exposed.
- The manufacturers enrich the device descriptions with semantic descriptions of device hierarchy and device topology. The device hierarchy information helps in configuration management. The topology information needs to be included in device description so that visualization can be re-configured dynamically.
- FASTory simulator tool is used as test bed for implementation. At the time of this thesis work, the RESTful web services for FASTory devices are still under developed. Hence FASTory simulator which works in the same way as FASTory line with advantage of exposing device description using RESTful service is considered for implementation.
- The orchestration system is considered only for scheduling resources and dispatching operations. The run-time support extended by ontology only during these operations is demonstrated.

1.4 Thesis outline

This thesis is organized in 6 chapters. Chapter 2 presents the theoretical background which describes the state-of-the-art technologies and tools related to the topic of thesis. Chapter 3 presents the research approach. Chapter 4 presents the ontology model and implementation of the approach on a use case system. Chapter 5 presents the discussion of the results obtained during the research. Finally, Chapter 6 presents the conclusion and future work.

2. THEORETICAL BACKGROUND

This chapter provides a review of the state-of-art technologies and concepts that are related to the context of this thesis.

2.1 Information and Communication Technology for Manufacturing Systems

Information and Communication Technology (ICT) has become a significant factor in contemporary factories. The use of ICT technology in factories has been increasing in the recent years offering IT support on all hierarchical levels of the factory. It increases the information flow across all levels of the factory. It is expected that ICT should be able to support the following objectives [5],

- Meeting the customer demands in terms of quality of the end products. This requires ICT to offer support on production processes to meet desired quality
- Speed and time for start-up of production system and for adapting to new technologies
- Lower production costs owing to motivation for adopting new technologies for production.

These objectives have been the factors for designing ICT in manufacturing domain. Moreover, the factories should be able to respond instantly to changing market requirements and flexible to adopt new technologies to thrive today's competitive environment. When the manufacturing system needs to adapt to new product variants, reimplementation of the shop-floor related IT might be necessary to keep the information system integrated and synchronized. The contemporary factories are also expected to provide extended visibility into all levels of the plant and capability to expose more information about different levels of factory so that there is availability of required information at right time to take right decisions.

Owing to these requirements in factories, IT support has increased in all hierarchical level of factory with increased information flow. The changing information architecture in manufacturing domain is represented using figure 1. It compares the automation pyramid of manufacturing enterprises to the new reference model of industrial information architecture described in [5] and [9].

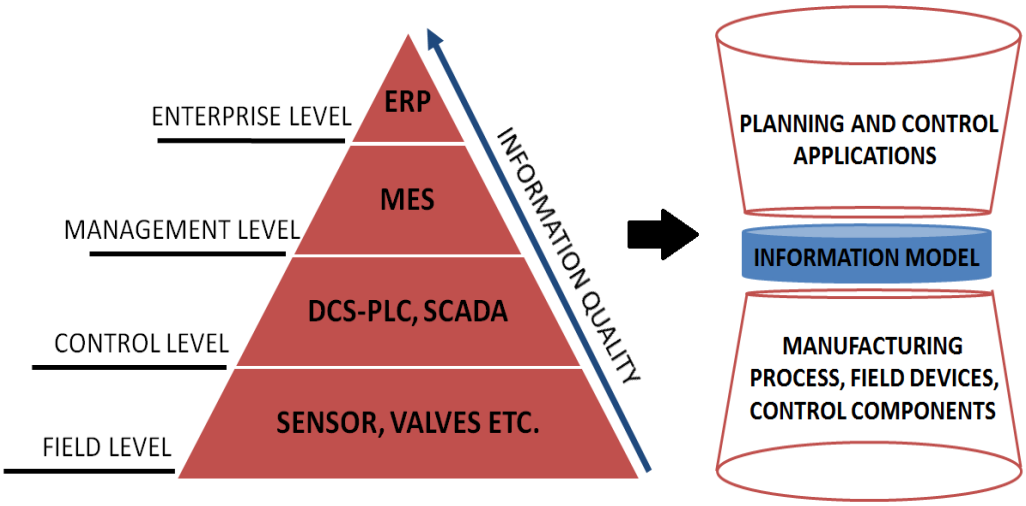


Figure 1.*Changing Industrial Information Architecture. Modified from [5]*

The automation pyramid is hierarchically layered and different levels represent similar functions. The shape of the pyramid highlights the decreasing amount of data from device layer toward the top layer for control of the production process. But the quality of information is less in device level. The advancements in the past 20 years, has changed the industrial information architecture. The new reference model described in [9] tries to depict the three dimensional integration which includes horizontal, vertical and life cycle of shop-floor device integration. The advancements in automation technology have made the field devices more intelligent with computational capabilities. It is seen as a vertical integration aspect. The distributed control paradigm has gained more importance and it has introduced modularization of the production process. Modularization is seen as a horizontal integration aspect. The architecture also integrates the life cycle of production process which affects the horizontal and vertical integration.

The advancements at field level and distributed control have opened doors for increased access to information in shop-floor and also for communication between the different levels of enterprise. The decision-making capability of manufacturing system depends on the availability of right data at right time. Hence the increased information access in factory shop-floor will support the efficient execution of the system. There is also increased accessibility to the monitoring parameters, which helps to build monitoring systems with interactive user interfaces synchronized with the system and capable to reflect the real-time changes in the system.

Moreover approaches like those involving the use of Service Oriented Architecture (SOA), helps to expose the information and functionality of the shop floor devices to information systems as services [10]. SOA is currently widely employed using Web Service (WS) technologies like RESTful services or WS-* protocols (SOAP Web Services) [11]. The exposed information from field level describes the field devices and

their configuration. These descriptions can be re-used to define new devices when they are added to the system. This increases the possibilities for dynamic re-configuration of the manufacturing systems.

Though the advancements in ICT have offered more visibility to shop-floor information, the major problem nowadays is handling the huge amount of information. The engineers are faced with the challenge to handle the massive information that flows between the enterprise layers. The required data has to be stored and has to be extracted for monitoring, analyzing and controlling the production process. Hence the need for knowledge representation and reasoning arises.

2.2 Knowledge representation

The knowledge representation and reasoning is a field of Artificial Intelligence (AI), which tries to describe the information about the world in certain formalisms that can be used by computer systems for solving complex tasks. Knowledge representation incorporates theories from psychology about how humans solve problems and uses it to represent knowledge. Reasoning can be defined as logic to automate the application of rules. It is described in [12] that knowledge representation and reasoning defines how an agent decides what it should do by using the knowledge of what it knows. The concepts involved and discussion on several formalisms are provided in [9].

In industrial automation domain, the knowledge representation is used to represent all the information about the system creating a manufacturing system model. It provides a means for storing the information about the system in both machine readable format and human readable format. Then this information can be used for the effective execution of the system. Several researches [13][14][15] have been done on using the knowledge models to improve the dynamic performance of the system. These research works puts forward the use of knowledge models together with other technologies to improve the flexibility of the manufacturing system.

Recent research works in automation domain incorporates the use of SOA together with knowledge representation to facilitate efficient management of manufacturing system information. An approach using the power of knowledge models and SOA control approaches to enable a fully open automated manufacturing environment is discussed in [15]. The approach allows the control to be configured by knowledge base automatically for specific manufacturing system. The implementation of knowledge-based industrial system with SOA to support run-time reconfiguration and adaptability of industrial system is demonstrated in [17].

Most of the research works in the recent years on knowledge-driven approaches have proposed the use of ontologies as a knowledge base. Ontologies allow describing the system in a hierarchical and organized manner. It also supports reasoning which enables

addition of new concepts to system model that are inferred by reasoning engines. This makes ontologies to update and extend at run-time automatically.

2.2.1 Ontology

In the context of artificial intelligence and knowledge representation, the ontology is best defined using the most cited definition provide by T.R. Gruber in [18]. He defines ontology as an explicit specification of conceptualization. The term ontology comes from philosophy where it refers to concept of existence. Hence the definition describes that ontology for AI systems can represent only the things that exist in the system. The other definition for ontology for semantic web is provided in [19] which states that "ontologies are data schemas, providing a controlled vocabulary of concepts, each with explicitly defined and machine processable semantics". A summary of other definitions for ontology from different perspectives is provided in [20].

The ontologies provide a hierarchical and well-structured format to represent the data. It provides a means to store, analyze, update and provide the knowledge of the system. This has led to the development of domain specific models. The domain specific models have formal representation of knowledge about a specific domain. A domain is defined as a particular area (e.g. manufacturing domain). In domain specific models, the concepts and relationships are represented in generic manner and the stored information is interpreted using reasoner.

Reasoning is the application of logic based on rules and relations of a set or sub-set. Reasoning allows for addition of new facts about the system which are concluded by reasoning machines based on rules. Hence new information can be generated at run-time and the ontologies can be updated dynamically. It can also have pre-defined rules that can be used by reasoning machines to classify and map the data accordingly.

The ontologies provide computer understandable definitions about the concepts involved in the domain and relationships among them. Moreover, it is also human-understandable. Ontologies are nowadays being widely used in different domains because they incorporate efficiency, flexibility and intelligence to the software system due to the following reasons outlined in [21],

- Ontologies allow the knowledge of a domain to be shared among people of software
- Ontologies separate domain knowledge from operational knowledge
- Ontologies make domain assumptions explicit
- Ontologies allow domain knowledge to be reused
- Ontologies also allow the domain knowledge to be analyzed

The ontologies are classified into different types based on the author. The classification of ontologies based on the subject of conceptualization is presented in [22]. Considering the semantics, [23] classifies the ontology based on rich and weak semantics. The ontologies with rich semantics can represent strong logical relations between the classes. Hence they provide more reasoning support.

2.3 Ontology languages

The ontologies are implemented using ontology languages. Choosing the right ontology language is crucial for modeling process. The required characteristics for ontology languages are expressiveness, inference mechanisms, ontology exchange, ontology integration, language integration for representing knowledge through web and existence of translators with minimum losses [21]. Different ontology languages for the semantic web is discussed in [24][21]. The classification put forward in [24] is based on syntax and classifies the ontology languages as traditional ontology languages and ontology markup languages (web-based language like SHOE, XOL, OIL, DAML+OIL). The ontology markup languages use a markup scheme for encoding the knowledge. The different ontology markup languages and their relations are shown in figure 2 [21].

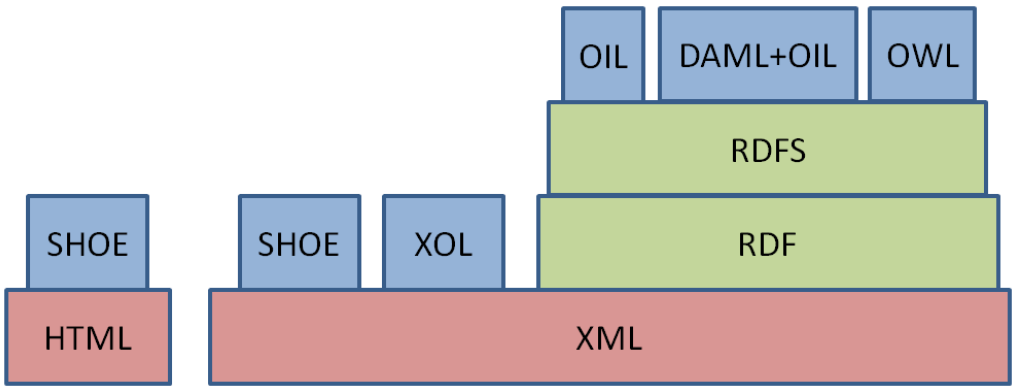


Figure 2. *Relationship of Ontology Markup languages [21]*

The ontology markup languages are based on Hyper Text Markup Language (HTML) and XML. Simple HTML Ontology Extensions (SHOE) was extended from HTML for defining semantic knowledge. Its syntax was also adapted to XML. All the languages that were developed later uses XML to encode the knowledge. eXtensible Out-of-band Language (XOL) evolved from OKBC1 protocol and used for exchanging formal knowledge models in the bioinformatics domain [25]. Later Resource Description Framework (RDF) was developed by W3C as a framework for describing the resources of the web. But RDF data models lack the definition of mechanisms to define relationship between properties and resources. For this purpose RDF Schema (RDFS) was developed which offers frame-based primitives for defining knowledge models. The languages which were developed as extensions to RDF(S) are OIL, DAML+OIL and

[1] OKBC Open Knowledge Base Connectivity is an API for accessing knowledge base stored in knowledge representation systems. For more information visit http://www.ai.sri.com/~okbc/

OWL. OIL was developed as an ontology language with rich semantics to offer rich reasoning support. DAML+OIL was also developed for the same purpose as OIL and allows semantic interoperability. The main difference between them is the extended expressiveness of DAML+OIL in representing individuals and datatypes [25]. The DAML+OIL forms the base for the development of Web Ontology Language (OWL) which is a language for processing the information on the web. It became the W3C recommended language from 2004. The main advantage of OWL is its increased expressiveness and semantics compared to other languages. It was designed not just to present the information to humans but to present machine-interpretable information to be used by applications for processing. This makes OWL superior to other languages and to be widely used in modelling manufacturing systems.

2.3.1 Web Ontology Language (OWL)

The OWL was developed to represent the information and present to applications in machine-readable format. OWL has greater expressivity and formal semantics. Hence it offers higher machine interpretability of web content that those offered by other languages such as XML, RDF and RDFS [26]. OWL includes three sub-languages which could be used based on the requirements of user. The sub-languages are OWL lite, OWL DL and OWL Full. Figure 3 which was referred from [21] shows the layers or sub-languages of OWL.

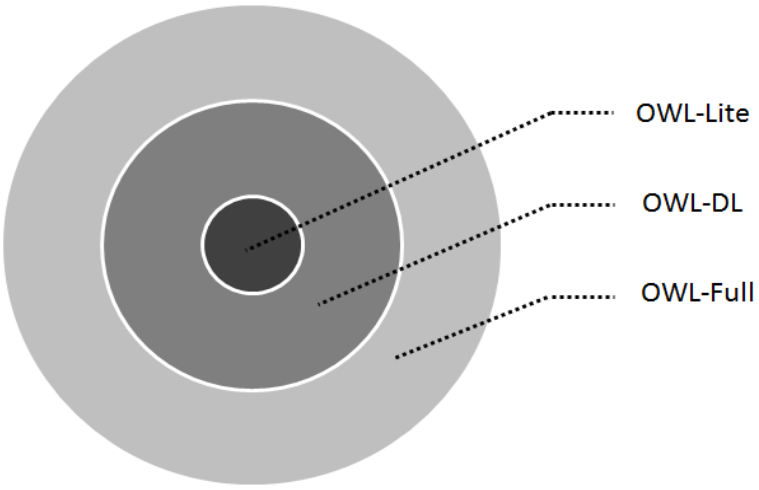


Figure 3. *Layers of OWL [21]*

OWL Lite is aimed for implementations which require primary features of ontology languages. It provides a classification hierarchy and simple constraint features. OWL DL is for implementations which require maximum expressiveness while still maintaining computational completeness and decidability of reasoning machines. It includes OWL Lite and is designed to support Description Logic business segment. OWL Full extends OWL DL and offers more increased expressivity at the expense of computa-

tional complexity. It is meant for implementations which require maximum expressivity with no computational guarantees.

The ontologies using OWL DL consists of three components which are classes, properties and individuals. The OWL ontology provides a hierarchical structure with superclasses and sub-classes. The classes are used to represent the concepts of the domain and the properties are used to provide relationship between the concepts. The individuals represent the objects in the domain of discourse. Classes could be considered as sets which includes these individuals. OWL DL provides 40 primitives, which includes 16 classes and 24 properties, to offer the required expressivity. The definition of these primitives is provided in [21].

In this thesis work, OWL is chosen as the ontology language, particularly OWL DL to model the manufacturing system because of the expressiveness and good computational capability/decidability offering greater reasoning support.

2.4 Ontology design methodologies

Ontology modelling of system involves many steps. In order to design an ontology model which could be adequate in representing the different aspects of a system, the need for a standard methodology arises. The methodology guides ontology developers by defining the steps to be carried out to design ontology following certain standards. In the past years, many research works has been done to develop methodology for ontology building. A methodology called METHONTOLOGY for building ontology from scratch and based on evolving prototypes is proposed in [27]. UPON (Unified Process for ONtologies) [28] is a methodology which is derived from the Unified Process (UP). It is an incremental and iterative methodology. Another methodology which uses UML and Business Process Modelling Notation (BPMN) for a dynamic ontology design is proposed in [29]. A detailed comparative study of some of the prominent methodologies is provided in [30], [31] and [32].

The main steps which can be found in the methodologies are shown in figure 4. Before building ontology, *feasibility study* is done to evaluate if ontologies are the right choice to be used in the particular application. Then the conventional steps of *domain analysis*, *conceptualization* and *implementation* are performed. The ontology development is supported using activities like *ontology reuse*, *knowledge acquisition*, *evaluation* and *documentation*. *Maintenance* and *use* are the activities which are performed after ontology development [33].

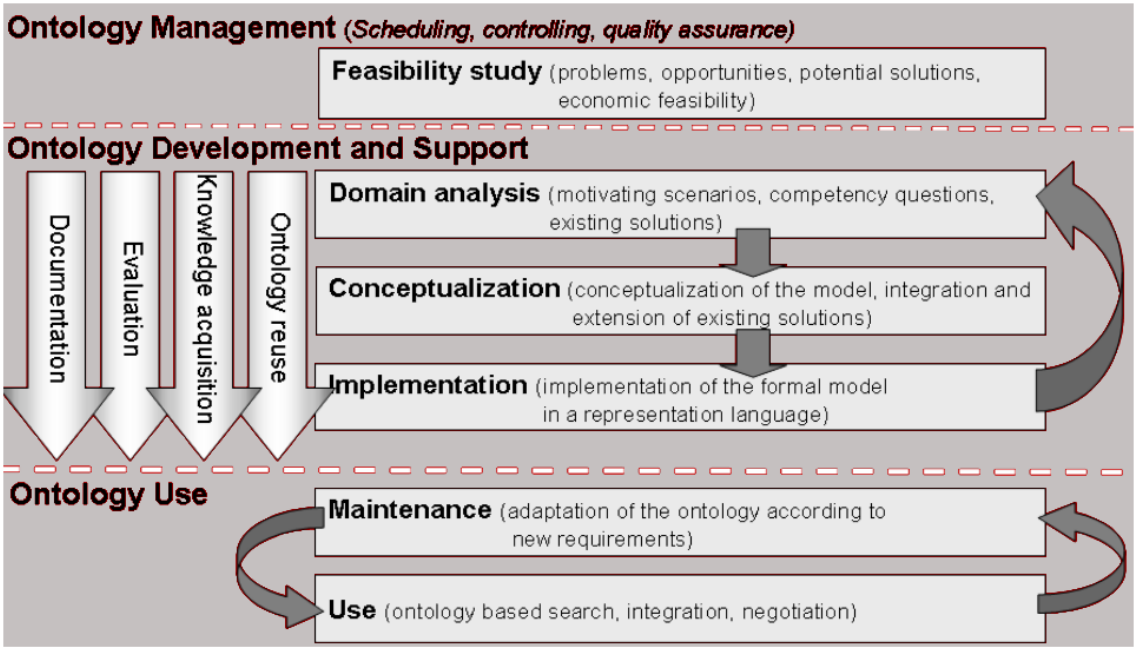


Figure 4. *General methodology for ontology development [33]*

There are also various specific methodologies developed for building ontologies for application in specific areas [34][35]. The methodology used mainly depends on the choice of ontology language. The methodology discussed in [21] provides a standard approach for ontology modeling with special focus on OWL DL language.

2.5 SPARQL query language

SPARQL (SPARQL Protocol and RDF Query Language) [36] is a semantic query language which is used for processing or manipulating the data stored in data bases. It has been a W3C recommendation since January 2008. It offers SELECT, CONSTRUCT, ASK and DESCRIBE query forms for retrieving the information stored in RDF graph stores. SPARQL 1.1 Update [37] was recommended by W3C in March 2013 and it is an update language for RDF graphs. It offers query forms to update, create and remove RDF graphs from graph store.

Since SPARQL supports manipulating of data stored in RDF format, it can be used with OWL ontologies. SPARQL has become a major technology for generating queries to retrieve, update, insert or delete the RDF data stored in OWL ontologies [38]. With SPARQL, the ontologies can be made up to date with system information at run-time.

2.6 Ontologies in Manufacturing Domain

Ontologies are increasingly used in manufacturing domain in recent years and have received increased attention due to the development of web-based solutions and semantic web. They provide the existing knowledge on the system concepts, entities and opera-

tions in a machine-interpretable format. These formal representations support the application of reasoning mechanisms for addition of new devices, re-configuration and solving other complex-problems. In this way, ontologies help to reduce the time and effort spent by the engineers on the laborious designing and configuration tasks.

The core concepts of manufacturing system are product, process and equipment. The ontologies which represent these core/upper concepts are discussed in [39]. The equipment is meant to offer a certain process and the product requires such process to be performed on it to transform to finished goods. This relationship among the core concepts is shown in figure 4.

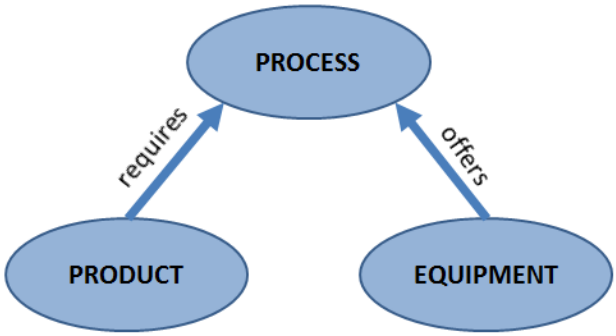


Figure 4.*Upper concepts of manufacturing system [39]*

There are different ontologies developed to represent these core concepts in different perspectives based on the needs of the system. A Product Knowledge Model (PKM) is an ontology developed in [40] to model the product information to facilitate knowledge sharing and re-use of information throughout product life cycle for product development among multidisciplinary organization. A meta-ontology using semantic TRIZ (Theory of Inventive Problem Solving) [41] is recently developed which enables innovative product designs to be applied to products of different domains. There is also many other ontology which are created to offer flexibility in product configuration based on customer demands [42][43].

The ontology developed in [13] models the basic concepts in a unit process and the relationship between these concepts. The manufacturing process usually involves a sequence of individual operations (assembly, drilling, etc.). By representing the knowledge on these individual operations or unit processes, ontology helps in process planning i.e. determining the sequence of unit processes to complete the entire process. One of the prominent ontology that models the assembly process design knowledge for assembly process planning is proposed in [44]. This ontology models the assembly requirements, spatial information, assembly operations and resources as important concepts. Moreover geometry and non-geometry information including assembly tolerances is also represented which were not considered in the previous models. This is shown in figure 5. The ontology can be further developed and extended to suit the assembly requirements of system.

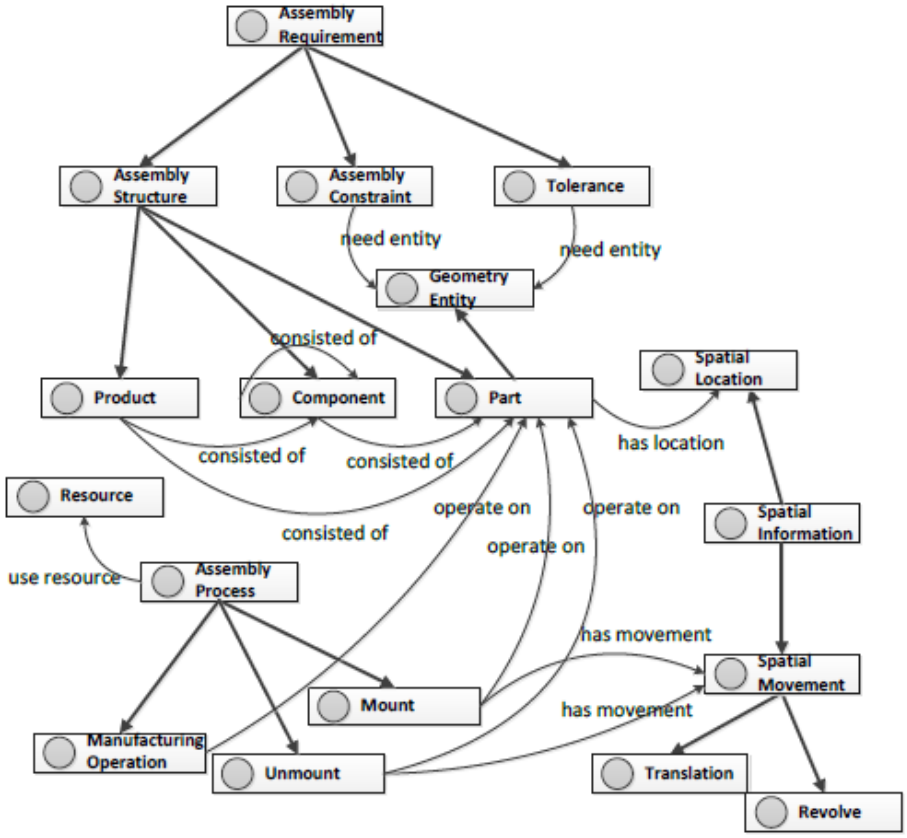


Figure 5. *Class diagram for Assembly Process Planning ontology [44]*

Most of the contemporary manufacturing systems are implemented using multi-agent and service-oriented approaches for achieving modularity, re-configurability etc. Ontologies are adopted in multi-agent systems for inter-agent communication and to extend support for production planning and scheduling. In service-oriented systems, ontologies are used to represent the system components (devices, equipment, etc.), control parameters, enterprise information etc., and offer them to other components of system as a web service. Ontologies used in some prominent and current approaches on multi-agent systems and service-oriented systems are discussed in below sub-sections.

2.6.1 Ontologies for Multi-agent systems

Holonic and multi-agent systems are based on decentralized control architecture and have been widely used for designing and implementing distributed and intelligent manufacturing systems [45][46]. A Multi-agent system is defined as a network of intelligent agents which are autonomous but cooperate and communicate with each other to perform tasks that are beyond their individual capability. Ontologies are mainly used in these systems as a knowledge base to support communication between the agents. The ontologies provide semantically enriched information enabling the agents to understand it in a broader context [47]. The set of standards for enabling interoperability of agents was developed by the Foundation for Intelligent Physical Agents (FIPA) [48] which

was established in 1996. The FIPA specifications recommended Knowledge Information Format (KIF) for ontologies for multi-agent systems. But KIF does not offer reasoning support and hence not able to offer automatic interoperability of agents [49]. The later developed approaches use ontology languages with reasoning support such as OWL for developing ontologies to provide interoperability among the agents.

ADACOR ontology is one that was created strictly based on theory of holonic manufacturing systems and tries to represent all the control features in a manufacturing system [50]. PABADIS is another ontology that models the resources, products and operations and it was used to develop reconfigurable manufacturing system [51]. The need for dynamic re-configurability with changing production requirements has become crucial to thrive today's competitive environment. Hence ontologies are developed to support reconfigurability in agent based systems. A similar approach is put forward in [52] and [53]. The Function-Behavior-Structure (FBS) ontology is used in [53] to derive the agent representation to support assignment of agents to processes dynamically. This ontology supports re-configurability of processes at run-time.

The ontology-driven approaches are used in multi-agent systems for real-time and distributed control of production systems. Modular, Intelligent and Real-time Agent (MIRA) proposed in the research work [54] uses the semantic knowledge represented in ontology about the agent's capabilities, tasks and surroundings and the reasoning mechanisms to generate IEC 61499 Function Blocks for controlling the production process. The proposed ontology-driven approach in [54] supports mass customization of products through the development of such agents enriched with semantic knowledge.

Ontologies are also developed to support resource scheduling which is one of the key concepts in dynamic manufacturing environment. Resource scheduling helps proper utilization of resources and minimizes the time and cost for production. The ontology proposed in [55], along with multi-agent architecture realized dynamic resource scheduling in manufacturing systems. The resources are represented in ontology as entities that provide certain operations. They are also related to an agenda which lists the unfinished assigned tasks. The agents use this knowledge along with dispatching rules to decide when and how the task will be performed.

The ontology put forward in [56] was developed to be used in the area of material handling especially for automatic parts transportation. Automatic transportation of material or equipment reduces material damage, cost and time of production. The ontology is modeled to reflect the relationship between Automatic Guided Vehicle (AGV) and transportation routes. The information stored in ontology is used along with shortest path algorithms to find transportation routes at run-time.

2.6.2 Ontologies for service-oriented systems

One of the main trends in today's research on manufacturing domain is to make all actors and field devices accessible from anywhere. Owing to the advancements in ICT, the research activities are proceeding in a direction to bring the features of field devices to cloud and it could be accessed from anywhere using common communication protocols. Among the research activities, one of the most employed research approach is Service-Oriented Cyber-Physical Systems (CPS) approach [57]. CPS is an engineered physical system with network of interacting elements with physical input and output. The Service-Oriented Architecture (SOA) plays a major role in the implementation of CPS.

SOA exposes the functionality of the devices of the system as services. The key advantages of SOA is its support for seamlessly horizontal and vertical communication, interoperability between components deployed in different network and offers reconfigurability to manufacturing system [58]. The combination of SOA and power of ontology facilitates effective management of system information offering run-time support for manufacturing system [17][59][60].

The ontology can describe the web services that are exposed by SOA about the processes offered by devices. Such ontology of services is used in SOCRADES (Service-Oriented Cross-layer Architecture for Distributed smart Embedded Devices)[61], an European project and other research works [60][62]. OWL-S (Semantic mark-up for web services), an ontology of services, is used to semantically describe the web services. Using the description of web services in ontology, the services can be dynamically discovered, composed and invoked.

An ontology model that represents the manufacturing semantics (i.e. production orders, processes, etc.) is presented in [14]. The ontology model facilitates run-time process information integration and update, enabling knowledge management using decision support applications. The expressivity of OWL was increased by applying Semantic Web Rule Language (SWRL) rules.

The ontology put forward in [58] models the production system and the information in the model is exposed using SOA (RESTful web services). The ontology describes the system components (i.e. equipment, sensor), orchestration related control parameters, and enterprise information (i.e. orders). It tries to model most of the important concepts in all levels of an industrial automation system. The class diagram of the ontology is shown in figure 6. This ontology can be generalized to be used with other production systems.

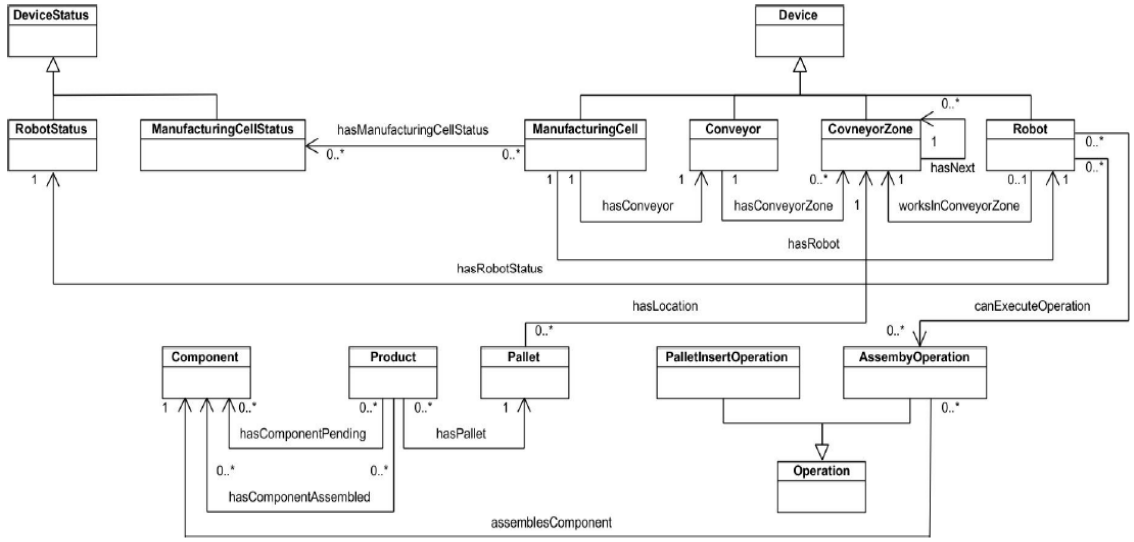


Figure 6. *Production line ontology class diagram [58]*

It is evident that most of the ontologies that were discussed in this chapter and other existing ontologies are upper ontologies (too generic) or too focused on specific areas of manufacturing domains. Hence the need for an integrated approach arises to combine the concepts in existing ontologies and add new concepts from different perspectives in order to provide ontology with broader capabilities. This thesis work aims to provide such an approach.

2.7 Visualization for Manufacturing Systems

Visualization of the manufacturing system plays a major role in controlling and monitoring applications. The visualization systems have evolved in the recent years to support user interactions and flexibility to reflect run-time changes in the system. The recent trend is to design visualization as web-based user interface owing to their flexibility. Some of the web-based Human-Machine Interfaces (HMIs) that are developed in agent-based and service-oriented systems are discussed in [63].The event-based and decoupled nature of SOA based shop-floor devices helps to develop visualization solutions that are easily configurable and integrated to the system. The visualization software system can easily interact and retrieve the information from the device, which is needed for visualization.

Owing to the increased accessibility to shop-floor information, the visualization solutions are expected to be completely integrated with this information for real-time control and monitoring purposes. One of the recent approaches is to design visualization using web-technologies. Designing HMIs as web application supports cross platform portability. The contemporary technologies like HTML5 (Hyper Text Markup Language 5), CSS3 (Cascading Style Sheets 3) and SVG (Scalable Vector Graphics) helps to create interactive web-based user interfaces. Moreover using technologies like Ajax (Asynchronous JavaScript and XML) helps to create asynchronous web applications i.e. the

application can send and receive data from server in the background without disturbing the web page. Thus by using modern web technologies, the visualization is provided as a web application which can interact with devices and provides the gathered monitoring parameters on web browser.

Although there are many advantages with these approaches, the visualization solution offered is not fully re-configurable and most of the screen elements are fixed. Only some details on the screen like monitoring parameters, animation of movement of pallets, etc., are updated dynamically. They do not support when new devices are added to system. In order to have a dynamic visualization which is completely integrated to system, re-configurable and easily maintainable, more information about the system and visualization screen is necessary.

The proposed approach in this thesis work tries to use the modern web technologies together with ontology as a source for visualization information which is needed to generate UIs. With ontologies, the visualization can be fully re-configurable and dynamic. Moreover, by representing the visualization information together with other information about system, uniform data representation format can be achieved.

3. METHODOLOGY

This chapter describes the ontological approach put forward in this thesis work. The technologies and tools used for the realization of the approach are described with reasons for their selection. The architecture of the system is presented first to describe the scope of the approach. Then the ontology design methodology for building ontology with broader capabilities and query patters for ontology reasoning are described.

3.1 System architecture

The ontology design approach was developed for systems implementing SOA. As discussed in section 2.1, the advancements in field level of factory during the past years have increased the access to information on factory shop-floor. This information is crucial for controlling and monitoring the execution of the manufacturing system. Service-oriented architecture design for performing software integration has received increased attention. It has become de facto in industrial informatics research [11]. SOA exposes the information and functionality of shop-floor devices as services. To realize SOA, web service technologies like RESTful services or WS-* protocols are widely employed [11]. It has been shown from several research works that SOA with web services technologies provides support for re-configuration, controlling and monitoring the system [10][65][66].

The exposed information from field devices through services describes the field devices and their configuration. These descriptions can be stored in ontology and re-used to define new devices when they are added to the system. This helps dynamic re-configuration of the devices in manufacturing system. The control parameters like the processes offered by equipment, current location of product, etc., can also be exposed using SOA and represented in ontology. These parameters are essential for the orchestration of processes. The orchestration systems can query the control parameters from ontology for run-time decision taking, scheduling processes, etc.

SOA also provides increased accessibility to the monitoring parameters. The monitoring parameters can be stored in ontology and updated with changes in the system. The monitoring systems can query this information in ontology to provide the monitoring parameters in user interfaces. This keeps the monitoring systems synchronized with real-time changes in the system.

The ontology presented in this thesis work was developed as a part of the research activity in eScop project. The architecture of the system which is put forward in eScop is

service-oriented and uses ontology-based knowledge management to achieve real-time control and improves the overall production control system architecture. The architecture of the system considered for the scope of this thesis work is presented in figure 7. It is composed of four layers

- Physical layer (PHL)
- Representation layer (RPL)
- Orchestration layer (ORL)
- Visualization layer (VIS)

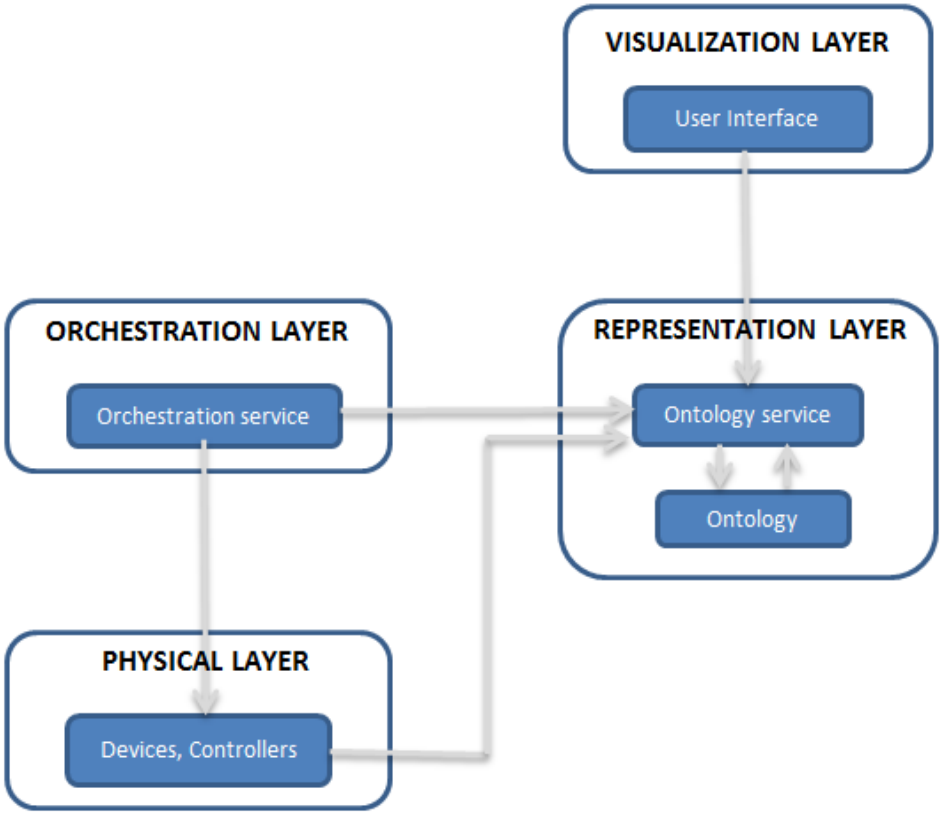


Figure 7. *System architecture*

The SOA encapsulates the functionality of each component and offers it as web services. The representation layer includes ontology and ontology service. The ontology service forms the central element of the architecture. The ontology represents all the information about the system so that the information can be exposed to other layers as services. The ontology services also allow orchestration service, physical devices and user interface applications to update the ontology based on the changes in the system. The physical layer includes the actual components of the system. The descriptions about the physical components such as their physical layout, services/processes offered, etc., are exposed to representational layer as services. It helps to update the ontology with the knowledge about physical components and their configuration. Orchestration layer is composed of orchestration engine to support orchestration of processes by taking run-time decisions. The ontology services offered by representation layer help orchestration

system for planning and scheduling the processes. The Interface layer composes user interfaces for visualizing and monitoring the system, or for requesting an order to be produced. The user interface can be re-configured dynamically based on changes in the system by using the visualization information in ontology.

The approach followed in this thesis involves design and implementation of ontology and ontology services for the above mentioned architecture. The ontology designed should be able to represent the different concepts needed to support the other layers. The main aim is to facilitate device configuration management (for Shop-floor devices), support for scheduling of processes (for Orchestrator) and dynamic configuration of user interfaces (for visualization). The developed ontology is generic and has broader capability for implementation in different types of manufacturing systems.

3.2 Shop-floor devices

The physical layer which is used for the approach is implemented using SOA. RESTful web services technology is one of the options to implement SOA. The industrial controller Inico S1000 can be used to connect to the sensors and actuators in shop-floor. It works based on scan cycle logic and uses RESTful service for exposing its functionality as web services. These web service operations can be invoked by the orchestrator to perform the desired actions on the physical devices.

The RESTful services can also be implemented to offer a subscribe mechanism i.e. the service requester can subscribe and receive notifications from the service provider. The POST method can be used to subscribe to the notifications. Thus the controllers can send notifications to other components of the system when certain events occur.

GET methods offered by RESTful service can be used to retrieve the information about device. This offers functionality to store the device information in ontology and can be updated based on changes in the system. In this way the device configurations can be managed with ontology.

3.3 Orchestration Service

The orchestration service is implemented based on the business logic for scheduling processes and dispatching by controlling the physical devices. The orchestration service gets information about Orders, processes offered by the device and device status etc., from ontology. Ontology provides the necessary knowledge to implement the business logic in ORL. Based on the information from ontology, orchestration service can invoke a sequence of web service operations provided by the controller and there by offering scheduling and dispatching functions. The orchestrator tool developed in the eScop project is used to support the implementation of the approach.

3.4 User Interface

The visualization layer of the architecture forms the front-end of the system providing a User Interface for monitoring the system. The ontology can represent the visualization information and provide to visualization application using ontology services. With the help of ontologies dynamically re-configurable UIs can be generated.

The visualization is designed as a web-application which provides the UI in the web-browser. The application is developed using technologies such as HTML5, CSS3 and JavaScript. AJAX is a group of web technologies (HTML, JS, XML etc.,) that allows the application to communicate to server in the background without interfering with the display of existing web page. Thus using this technology, the information in ontology can be retrieved in the background and visualization can be updated. The visualization application retrieves from ontology the information about screen composition, topology and symbols. This information in ontology alone is not sufficient to generate visualization and some additional metadata is necessary. In order to visualize a shop floor layout, the arrangement/position of the graphical elements in screen is needed. Hence the visualization implements layout algorithms to position the elements in screen based on the topology information. Moreover it is not possible to store in ontology, the starting element of the layout, orientation, etc., if they change with different screen implementations and interests of the user. These metadata can be provided by user during the configuration phase of visualization screen. Hence visualization configuration interface is provided first with layout algorithms to automatically determine the positions of screen elements. It can also obtain the position and other metadata from users. The metadata provided in the configuration screen is then saved to ontology and can be used by the visualization application to generate visualization. When there are changes in the system at run-time, the visualization configuration interface can be re-configured easily with the help of ontologies and there by supporting dynamic visualization.

3.5 Language and Support tools for Ontology

In this thesis work, the ontology is designed using OWL DL language. OWL DL is chosen as the ontology language, because it offers more expressiveness. At the same time it also provides good computational capability/decidability which greatly supports reasoning. The ontology editor used to design and develop ontology is Olingvo [67]. Olingvo is a graphical application to define ontologies developed in Tampere University of Technology, FAST laboratory. It supports building and editing OWL models, SPARQL querying for extracting and updating the information in OWL models and SWRL rules for expressing the rules and logic needed for reasoning.

The SPARQL is decided to be used as the query language for providing the required ontology reasoning. As discussed in section 2.5, SPARQL is one of the major technologies for generating queries for OWL ontologies. It offers query forms for creating, up-

dating and removing information in OWL models. SPARQL is also supported by Olingvo. For this reasons, SPARQL is chosen as the query language.

3.6 Ontology Design Methodology

The ontology modeling is performed by following a design methodology. The methodology put forward in the thesis involves eight steps guiding the design process of ontology and it is an iterative process to improve the ontology. Based on the application, ontology will be refined to suit the needs of the system. Using this iterative process, ontology can be improved over time to be used in different types of manufacturing system with SOA. The ontology design steps are shown in figure 8.

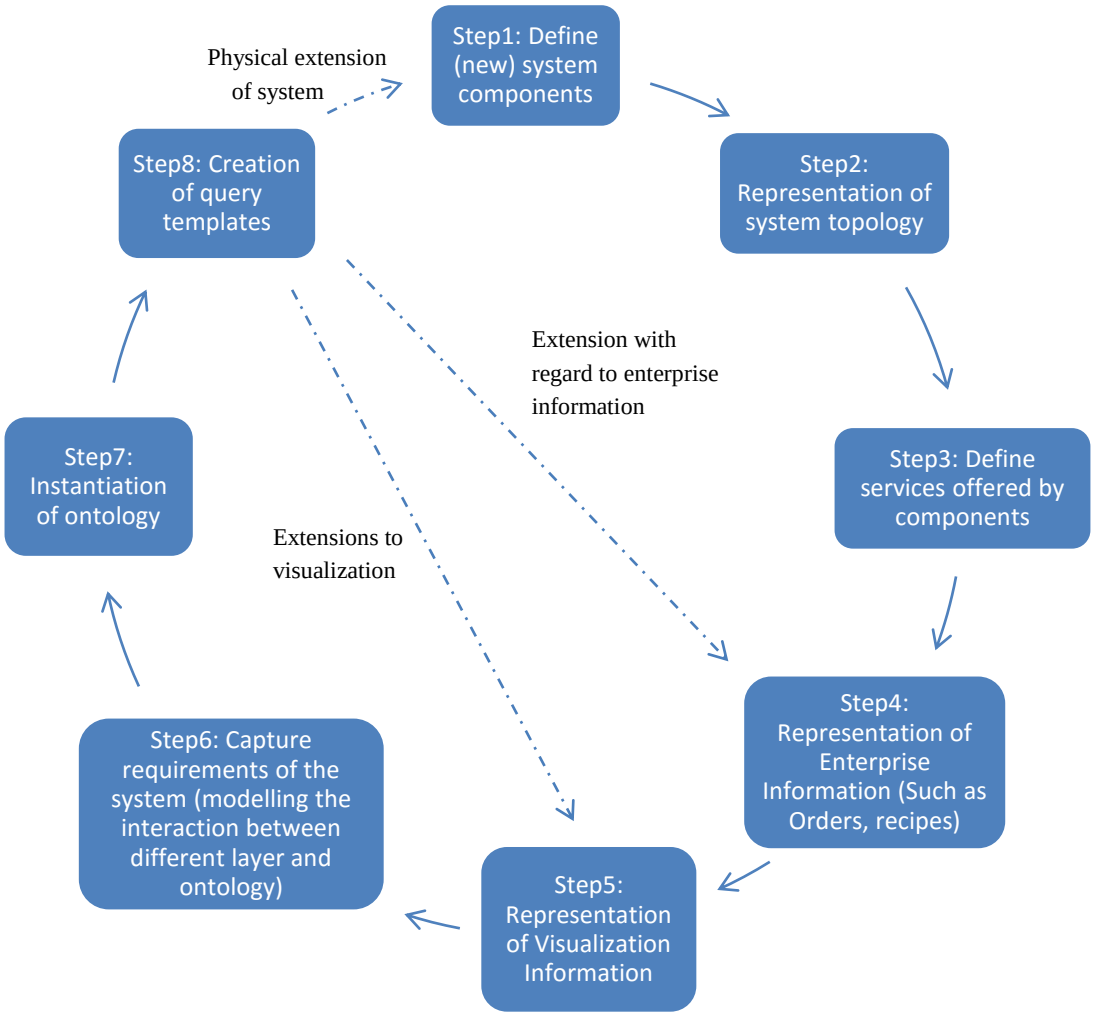


Figure 8. *Ontology design methodology*

The design steps of the ontology are aimed at representation of concepts involved in different layers of the architecture presented in section 3.1. The various concepts that can support the functionality of the shop-floor devices, orchestration engines and user interface are identified and modeled in ontology. The step1 to step5 helps to develop a

more generic ontology model and steps 6 to step 8 depends on the specific system where the ontology will be implemented.

The first step in the design process is to define the system components. The components are units that compose the system. It can be a device or equipment, industrial controller, sensor, conveyors etc. Representation of the components in ontology provides a way to describe the system. It is one of the basic design steps in any manufacturing system ontology. By representing the components, system configuration can be stored in ontology. It also helps to manage the configurations at run-time, by re-using the representation to define new components. For modeling the knowledge about the components, following questions are considered.

- How the component can be identified? There may be more than one component which belongs to same class. E.g. *conveyor1* and *conveyor2* both belongs to *Conveyor* class. In order to distinguish them, unique identifiers can be defined in ontology.
- What is the component type? The system is considered to be composed of various components such as conveyors, sensors, processors, etc. which could be represented as sub-classes. The components of the same class can be of different types e.g. *sensor1* is a *temperature sensor*, *sensor2* is a *barcode reader*.
- What the component is composed of? A component can include other components. E.g. *conveyor1* includes *sensor1*.
- Is it a part of another component or sub-system? A component can also be a part of another component. E.g. *conveyor1* belongs to *workstation1*

By answering the above questions, the knowledge about the system component is achieved and can be used to model the component.

The second step is to define the topology of system components. The representation of topology helps to understand the orientation of the components in shop-floor. The topology can be represented by defining where the components are physically present with respect to its neighboring components. The topology information can be useful for visualization applications to understand the shop-floor layout structure.

The third step is to define the services that are offered by the components. In a service-oriented architecture based system, the components expose their functionality or processes as a service. To know what component offers what processes, the component needs to be mapped with corresponding services. This information can help orchestrator to schedule the processes. Hence the representation of the services and their mapping to components are performed in this step.

The fourth step involves representation of enterprise information that is needed to support the orchestration process. The information about orders and recipes are represented

in ontology. The order details like the product type to be produced, quantity, recipe, etc., are modeled to represent the order. The recipe is represented as the operations to be performed on each product. The order and recipe representation helps to store information that can support orchestration engines to plan and schedule the processes.

The fifth step is the representation of visualization information i.e. information about the graphical elements of visualization screen. It includes

- What are the graphical elements present in the screen?
- How the graphical elements relates to system components?
- What is the composition of each element?
- What is the position of the graphical element on screen?
- What symbols should be used to represent the graphical elements?

By representing the visualization information in ontology, visualization can be made dynamic. The information can also be re-used to define graphics on screen when new components are added to system. Thus this step aims to represent the visualization information which can support dynamic re-configuration of user interfaces.

The sixth step is capturing the requirements of the system where the ontology will be implemented. For this the interactions needed between different layers such as PHL, ORL and VIS with ontology should be modeled. Modeling the interactions helps to define what information needs to be stored in ontology and presented to other layers. UML Sequence diagrams can be used for modeling the interactions between different layers and ontology.

Based on step six, the required instances are created in step seven. The final step in the methodology is creation of query templates. Queries provide the required reasoning for ontologies. It serves to retrieve or manipulate the information in ontologies. The use of SPARQL, semantic query language, for generating queries for OWL ontologies was discussed in section 2.5. With SPARQL, query templates can be created to retrieve, update, create and delete the RDF data in ontology. The SPARQL queries are created and tested using Olingvo, ontology editor, before they are implemented. The required templates are clear from step six as they serve the interaction needed between different layers and ontology. The query templates are created using certain patterns which are discussed in section 3.7

Thus by following the eight steps of methodology, ontology is designed. It is developed to represent the different concepts that can support the different layers of the system architecture. The iterative design helps to go from step8 to step1 in case of expansion of system with regard to its physical components. The methodology also allows going directly from step8 to step4 when there is expansion of system with regards to only enterprise information. Then the new information can be represented in ontology. Likewise, it also allows going from step 8 to step 5 when there is new extensions to visualization.

The overall iterative process helps to refine the ontology over time for implementation in different types of manufacturing systems.

3.7 Query Patterns

Some of the query patterns which can be used for creating templates are discussed below. The patterns are based on query forms such as SELECT, DELETE and INSERT. A simple example ontology which is given by the class diagram in figure 9 is considered to describe the query patterns.

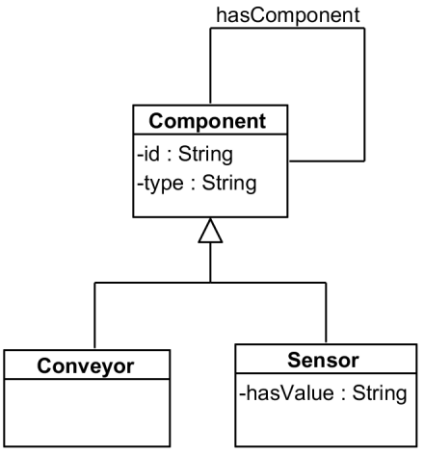


Figure 9. *A simple component ontology example*

A basic query pattern is shown in figure 10. It uses SELECT query form to retrieve all the triples in the ontology and provides a dataset table with *subject*, *predicate* and *object* which describe the relationship of all ontology components.

```
SELECT ?subject ?predicate ?object
WHERE {
?subject ?predicate ?object
}
```

Figure 10. Basic query pattern

When this basic query is applied to the ontology given in figure 9, the resulting query solution can be as shown in table 1. The table 1 shows only some example matches and there can be more depending on the use case.

Table 1. Query result for basic query pattern

Subject	**Predicate**	**object**
conveyor_1	hasId	id_1

conveyor_1	hasType	type_1
conveyor_1	hasComponent	sensor_1
sensor_1	hasValue	0

The queries also have predefined namespace prefixes for the URIs. For retrieving the information about a particular *subject***,** the predicates are defined separately in the query based on what information (*object*) is needed. This type of query pattern is shown in figure 11. The *id* and *type* of *conveyor_1* is retrieved in this example. This query pattern can be used to retrieve the necessary information about any *subject.* Hence this type of pattern is useful for orchestrator to get information about the devices which is needed for planning and scheduling the processes. The same pattern can be used for getting information about graphical elements for visualization.

```
PREFIX comp:<http://www.escop-project.eu/MSO.owl#>
SELECT ?id ?type
WHERE{
comp:conveyor_1 comp:hasId ?Id;
                comp:hasType ?type.
}
```

Figure 11. *Query pattern for retrieving information*

The example query pattern for inserting data in ontology is shown in figure 12. This query creates a *conveyor_2* instance for Conveyor class and inserts the information such as *id* and *type* for *conveyor_2*. This query pattern uses RDF primitive for specifying the class type where the subject should be instantiated. For using RDF primitives, first the rdf prefix is included in the query. The datatype objects that need to be inserted for the specific subject, is directly included in the query with corresponding predicates. This type of query pattern is helpful for inserting the devices and its information in ontology at the time of initialization of the system. Moreover, during run-time new devices, orders etc. can be added to the system. Hence the physical layer services and visualization applications can insert the run-time information to ontology by using this query pattern.

```
PREFIX comp:<http://www.escop-project.eu/MSO.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
INSERT DATA{
comp:conveyor_2  rdf:type  comp:Conveyor;
                 comp:hasId  "id_2";
                 comp:hasType  "type_2".
}
```

Figure 12. *Query pattern for inserting data*

The data is inserted also by matching the patterns in WHERE clause. This type of pattern uses INSERT query form and can be used to add information to existing subjects in ontology. An example for this case can be inserting a new sensor under existing conveyor. The query pattern for this example is shown in figure 13. Depending on the complexity of the concepts in ontology, the pattern in WHERE clause can also be complex. This query pattern can be used for inserting recipe to existing orders, inserting topology to existing system components etc. Thus the orchestration service and physical layer services can insert data to ontology using this pattern.

```
PREFIX comp:<http://www.escop-project.eu/MSO.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
INSERT {
comp:sensor_2 rdf:type comp:Sensor.
comp:conveyor_1 comp:hasComponent comp:sensor_2.
}WHERE{
comp:conveyor_1 a comp:Conveyor.
}
```

Figure 13.*Query pattern for inserting data with pattern matching in WHERE clause*

The query pattern example for deleting the information in ontology is shown in figure 14. The example is used to delete the *sensor_2* instance of Sensor class and removes all the relations that involve *sensor_2*. With this query pattern an object can be deleted from ontology along with all the relations concerned with that object. This type of query patters is useful for maintaining the configuration of the system. When a device is removed from system at run-time, the representation of the device and its mapping can be deleted from ontology. It is also useful for orchestrator since some information needs to be deleted in ontology during the orchestration process. For example the orchestrator can maintain a recipe list of the processes yet to be performed on the product and it may require deleting the particular process on its completion.

```
PREFIX comp:<http://www.escop-project.eu/MSO.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
DELETE{
comp:sensor_2 rdf:type comp:Sensor.
?subject ?predicate comp:sensor_2.
}WHERE{
comp:sensor_2 a comp:Sensor.
?subject ?predicate comp:sensor_2.
}
```

Figure 14. *Query pattern for deleting information*

The most used query pattern for manipulating the data in ontology is for updating the existing information based on changes in the system. Such a query pattern example which makes use of both INSERT and DELETE query forms is shown in figure 15. The query pattern in WHERE clause tries to match the patterns in ontology. The matching objects / datatypes are deleted with DELETE clause and new objects/ datatypes are added with INSERT clause. The orchestration service uses this pattern for example to up-

date the ontology about the status of devices. Similarly the visualization application can also use this pattern to update visualization parameters.

```
PREFIX comp:<http://www.escop-project.eu/MSO.owl#>
INSERT {
comp:sensor_2  comp:hasValue 1.
}DELETE {
comp:sensor_2  comp:hasValue ?value.
}WHERE{
comp:sensor_2  comp:hasValue ?value.
}
```

Figure 15. *Query pattern for updating information*

The patterns discussed in this sub-section are simple and based on the application it can be more complex. SPARQL also provides patterns like Distinct, Limit, Filter, Reduced, Order By, Offset, Projection etc., [37]. These patterns can be used to modify the sequence of the solution. Moreover keyword like OPTIONAL can be used for specifying the optional parts of the patterns and UNION can be used for matching the alternative patterns.

3.8 Ontology services

The ontological approach put forward in the thesis uses ontology services to exposes the knowledge in ontology to other layers such as PHL, ORL and VIS. Ontology services provides service to query and update the information in ontology. Hence it acts as integrator node for all layers.

The ontology services are created based on the interaction needed between different layers and ontology. The ontology services can be implemented using RESTful web services to realize the service-oriented architecture. The RESTful web services can be developed in a JAVA programming environment. The services needed for different layers can be provided using REST controllers. It provides GET, POST and DELETE methods for interaction with ontology. Each method defines a specific service route. It uses the respective query templates for retrieving, updating/inserting or deleting the information in ontology. To retrieve information from ontology GET request methods can be used. The information retrieved can be composed in JSON format before the response is sent back to the respective layers. JSON format can be used since it is a light weight data exchange format.

The parameters to be inserted or updated in ontology can also be passed for the service using JSON format through POST request methods. The parameters passed in JSON format can then be updated in the query template to generate queries for inserting/updating the parameters to ontology.

4. IMPLEMENTATION

This chapter first describes the design of ontology with broader capabilities to support dynamic configuration management and execution of manufacturing system. The ontology is developed using Olingvo, ontology editor. Then the description of the Use case system and the realization of the implementation scenario which tests the capability of the ontological approach are presented. The tool for exposing and manipulating the information in ontology through services is developed using Java programming language in Netbeans IDE.

4.1 Manufacturing System Ontology (MSO)

Olingvo, ontology editor is used for the ontology designing process. The step1 to step5 of the methodology presented in section 3.6 is used to build the ontology which is generic to be implemented with different systems. The class diagram and property diagram of the ontology is presented in figure 16 and figure 17. The description of the ontology designed is discussed in detail in below sub-sections.

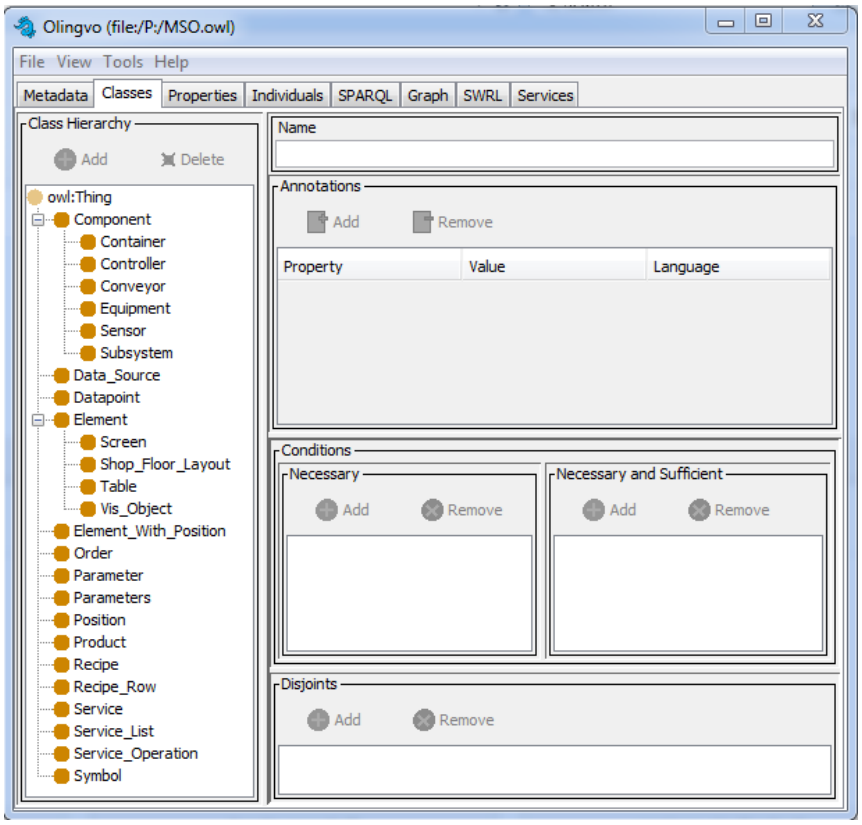


Figure 16. *Screenshot of class hierarchy in ontology*

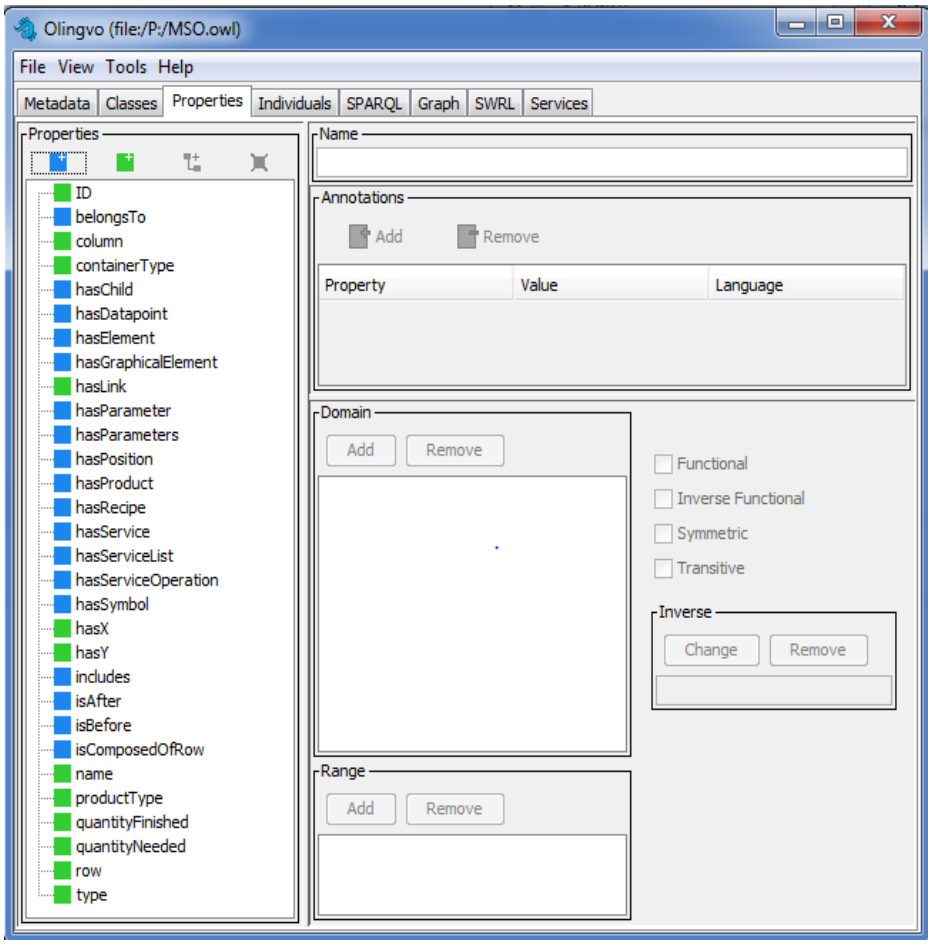


Figure 17. *Screenshot of properties in ontology*

4.1.1 System components and topology

The first step in ontology design methodology is modeling the system components. The component can be an equipment, industrial controller, sensor, conveyors, containers (i.e. case, pallets etc.) and sub-systems. A *Component* class is created in ontology to represent these physical units. It includes sub-classes like *Equipment*, *Controller*, *Sensor*, *Conveyor*, *Container* and *Subsystem.* To have a unique identifier for the components, a data property *ID* is provided. The type of the component is described by using another datatype property *type*. A component can be composed of other components or can be a part of another component. This association between the components is established by object properties *includes* and *belongsTo* respectively. Some examples for expressing these relationships can be *Conveyor_1* ***belongsTo*** *Sub-system_1*, *Subsystem_1* ***includes*** *Equipment_1*, etc.

The properties *includes* and *belongsTo* express self-relationship for *Component* class. The class diagram of the portion of ontology representing the components and their relationship is shown in figure 18. This portion of the ontology represents the physical

units that compose the system. It is especially useful in storing and maintaining the system configuration. When there are changes to system in run-time for e.g. new equipment (*Equipment_2)* is added to a sub-system (*Subsystem_2)*, it can be instantiated under *Equipment* class and its relationship with the sub-system can be provided using *belongsTo* property dynamically. Thus this portion of ontology serves to support management of system configuration.

The second step in ontology design is the representation of topology of system components. It is the knowledge about where the components are physically located in the system. The topology information is essential for understanding the system layout and to visualize the system. The monitoring applications should provide the visualization of shop-floor layout for control operators or operator managers to monitor the system and take decisions. To define the topology, two object properties are introduced in ontology; *isAfter, isBefore*. These properties define what components are present after a component and before a component. For e.g. the representation *Conveyor_2* ***isAfter*** *Conveyor_1, Conveyor_2* ***isBefore*** *Conveyor_3*; defines the location of *Conveyor_2* in between *Conveyor_1* and *Conveyor_2*. Using the *isAfter* and *isBefore* properties the monitoring applications can obtain knowledge about the system layout and can use it for generating visualization. The class diagram of the portion of ontology representing the system components is shown in figure 18.

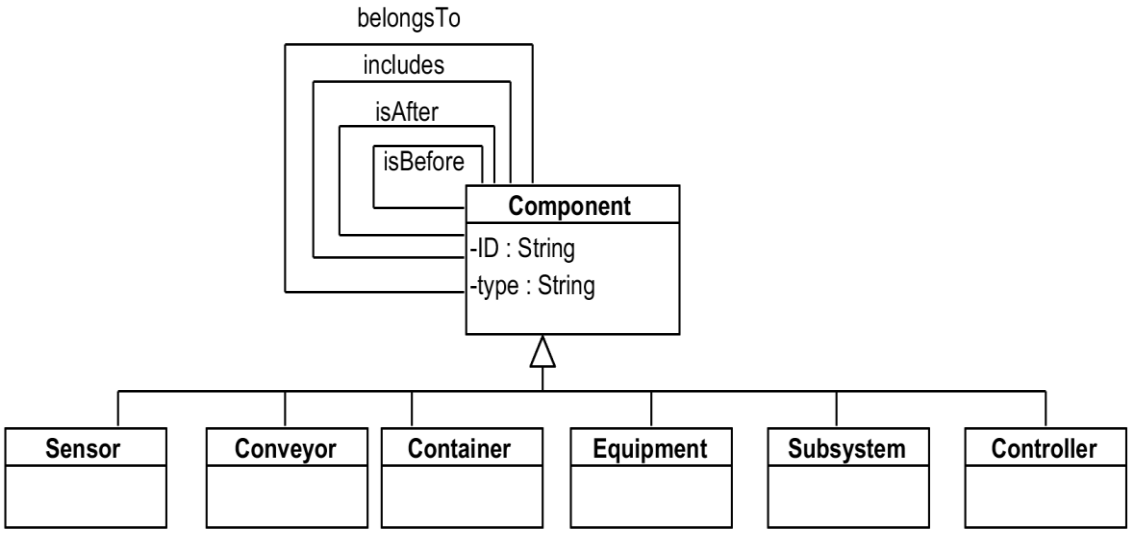


Figure 18. *Class diagram for representation of System components*

4.1.2 Services

The third step in ontology design is modeling the services offered by components. Following SOA, the system components expose the operations performed by them as services. The information on what components provide what services should be presented in ontology to facilitate orchestrators to plan and schedule the processes. For this pur-

pose, *Service_List* class is provided in ontology. It represents a list of services provided by a particular component. The *Component* class is linked to *Service_List* class using *hasServiceList* property.

The *Service* class is provided to represent the individual services. The *Service_List* class is linked to Service class with property *hasService*. Each service has a name given by datatype property name and involves an operation given by *hasServiceOperation*. The *Service_Operation* class represents the operation provided by the component upon invocation of the service. The service operations have unique id given by *ID* property and provides the service URL link (using datatype property *hasLink*) from where the service can be invoked. The portion of ontology which includes representation of services is shown in figure 19.

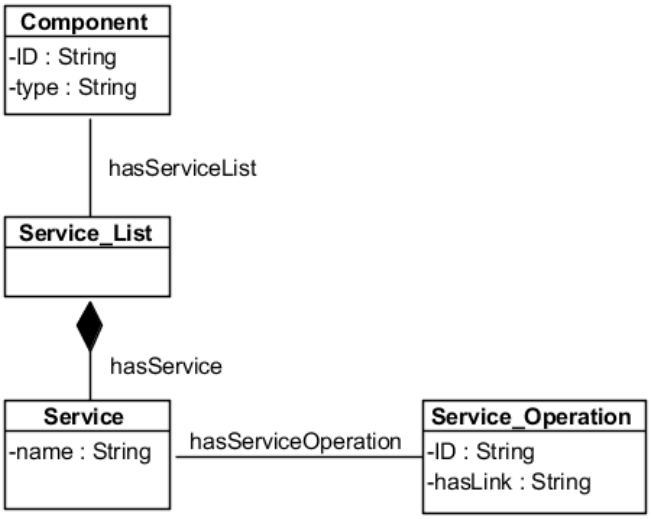


Figure 19. *Class diagram for representation of Services*

An integrated approach is followed to design the ontology by bringing together the different concepts in manufacturing system and concepts involved in controlling and monitoring the execution of system. The ontology presented is more generic to suit the implementation for different manufacturing systems. Based on the implementation, it can be easily extended to suit the requirements of the system.

4.1.3 Enterprise Information

The fourth step in the ontology design is to represent the enterprise information. The enterprise information such as orders and recipes are vital for supporting the execution of system. Recipe defines the processes to be performed on each product to transform to finished product. The information on orders and recipes are essential for orchestrator to plan and schedule the processes accordingly. For this purpose, they are represented in ontology. The order information should include definition of product, quantity of the

product needed and recipe. Recipe information should include the list of processes to be performed on each product.

Order class is created in ontology to represent the orders. The orders are identified by unique identifiers given by data type property *ID*. The datatype property *productType* is used to define the type of product to be produced. The datatype property *quantityNeeded* is provided to store the quantity of products to be produced. To define the quantity of finished products, datatype property *quantityFinished* is provided.

Product class in the ontology represents the products. Products are also identified with unique identifier given by datatype property *ID*. The datatype property *type* defines the type of the product. Each product is related to some order and this relationship is given by the object property *hasProduct*.

The order is also related to recipe as it defines what processes are to be performed to produce the product. For this purpose, *Order* class is related to *Recipe* class using the property *hasRecipe*. Recipe also has an id given by property ID and it includes recipe rows represented by Recipe_Row class. Each recipe row represents a process to be performed on the product. There can be more than one parameter defining a particular process and these parameters can be defined under a recipe row. The parameters depend on the type of system and product requirements. Based on the system where the ontology will be implemented, more properties can be created to represent the parameters. In this way, ontology can be extended to suit specific requirements based on implementation. Also the representation provided in the ontology includes orders with only one product type and it can be extended to represent orders which include more than one product type. The portion of the ontology which represents the enterprise information is shown in figure 20.

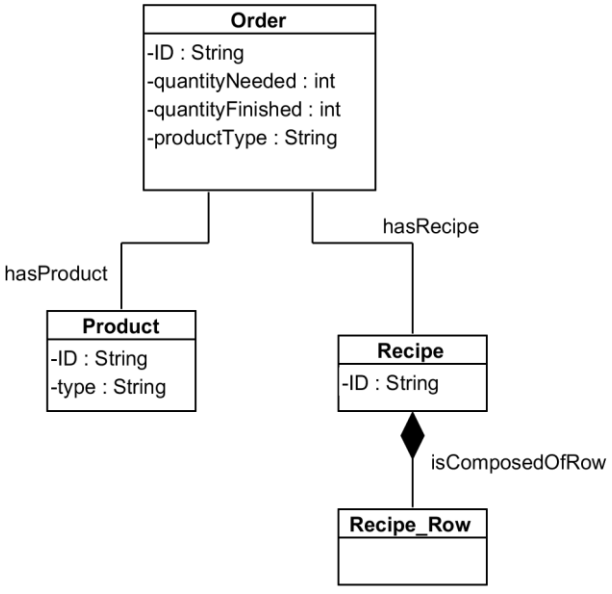


Figure 20. *Class diagram for representation of Enterprise Information*

4.1.4 Visualization Information

The fifth step in the ontology design is to model the visualization information. Visualization of the manufacturing system is crucial for controlling and monitoring applications. To have a dynamically re-configurable visualization, the visualization information is represented in ontology. This information can be presented to visualization applications for creating UIs. The information can also be re-used to define new screen elements at run-time providing a completely re-configurable visualization.

The visualization screen usually consists of graphical elements like shop floor layout, graphs, tables, maps, buttons etc. To represent screen composition in ontology, classes and properties are created to define graphical elements and their relationship. The screen itself can be considered as a graphical element and it includes other elements like shop floor layout, table, graphs, data points, etc. The *Element* class in created to hold these graphical elements. It includes sub-classes like *Screen* class, *Shop_Floor_Layout* class, *Vis_Object* class and *Table* class. Based on the needs of the screen, the Element class can be extended with other sub-classes.

The composition of the elements is represented by *hasChild* property. The *Element* class should have a self-link since an element might be composed of another element for e.g. screen can be composed of shop floor layout whereas shop floor layout can contain conveyor symbols. However direct representation of self-relation in Element class is avoided here since there might be representation of same information in different screens. Ontology must support reusability of the graphical elements in various screens but still adjusting to the requirements of such screens. One such issue is seen in defining the position of the elements in screen. For example an operator screen element with table element at some position on the screen. The same element can be in a different position in another screen. To avoid duplication of elements, another class *Element_With_Position* is defined. *Element* class is linked to *Element_With_Position* class using *hasChild* property and *Element_With_Position* class is linked to *Element* class and *Position* class using *hasElement* and *hasPosition* property.

The positions are defined based on container type of parent element. For example an element with container type canvas has its child element at position defined by x,y co-ordinates. But if the container type is table then the child position is defined by rows and columns. The *containerType* property is created to know the container type so that positions can be defined based on it. Once the classes and properties are created to represent screen composition, the next step is to relate them with the system in order to have visualization integrated with the system.

The graphical elements must be synchronized with the real system to represent the changes in system for monitoring purposes. If ontology is synchronized with the system, any changes in the real system, updates the ontology and this change can be re-

flected in the visualization. For this purpose an object property *hasGraphicalElement* is created to link the system components like transporter, sensor etc., to the graphical element. This is shown in figure 21.

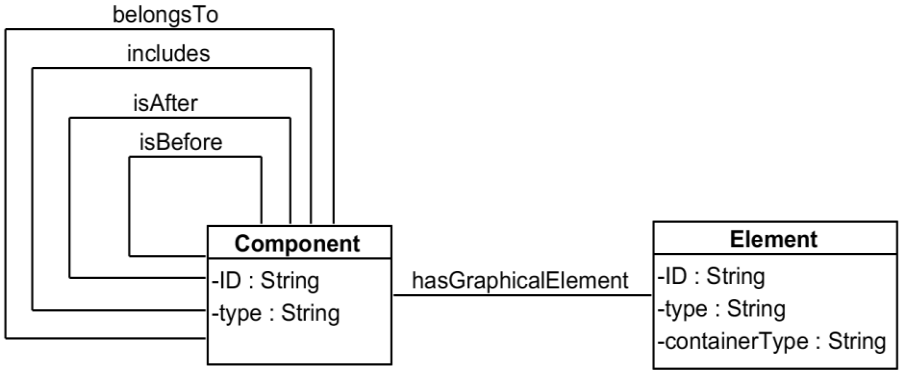


Figure 21. *Mapping between System Components and Graphical Elements*

The portion of the ontology discussed in section 4.1.1, represents the system components and their composition. It also defines the physical location/arrangement of the components. This knowledge about the components in shop floor is required for the visualization of shop floor layout. Since the *hasGraphicalElement* property links the graphical elements and system components, this information can be accessed easily to present the layout structure in visualization display.

The graphical elements are visualized as symbols in the UI. *Symbol* class is created in ontology to represent the symbols and *hasSymbol* property relates the elements and symbols. These symbols may range from simple symbols like circle,rectangle, etc. to complex symbols (e.g. conveyor symbol which is a combination of many simple symbols). The symbol templates for graphical elements can be created and stored in visualization applications. But mapping between the graphical elements and templates is necessary for the application to decide the symbols for visualization. For this purpose *hasVisTemplate* property is provided. The symbol parameters like orientation, height, width, etc. can also be provided in ontology using *Parameter* class. The name of the parameter and its value can be stored using *hasName* and *hasValue* property respectively. Each symbol can have more than one parameter. Hence the parameters are listed by using *Parameters* class. The symbols (e.g. conveyor symbol) may also consist of parameters like sensor values for monitoring purposes. In this case, the parameters should be mapped to the values from data sources (e.g. sensors). *Datasource* class (sub-class of *Datapoint*) is created to represent the data point from where the visualization parameter can be obtained. The relationship between parameter and data source is given by *hasDatapoint* property. This helps to obtain the parameter value from the source and present it on visualization screen.

The class diagram of the portion of ontology representing the visualization information is shown in figure 22.

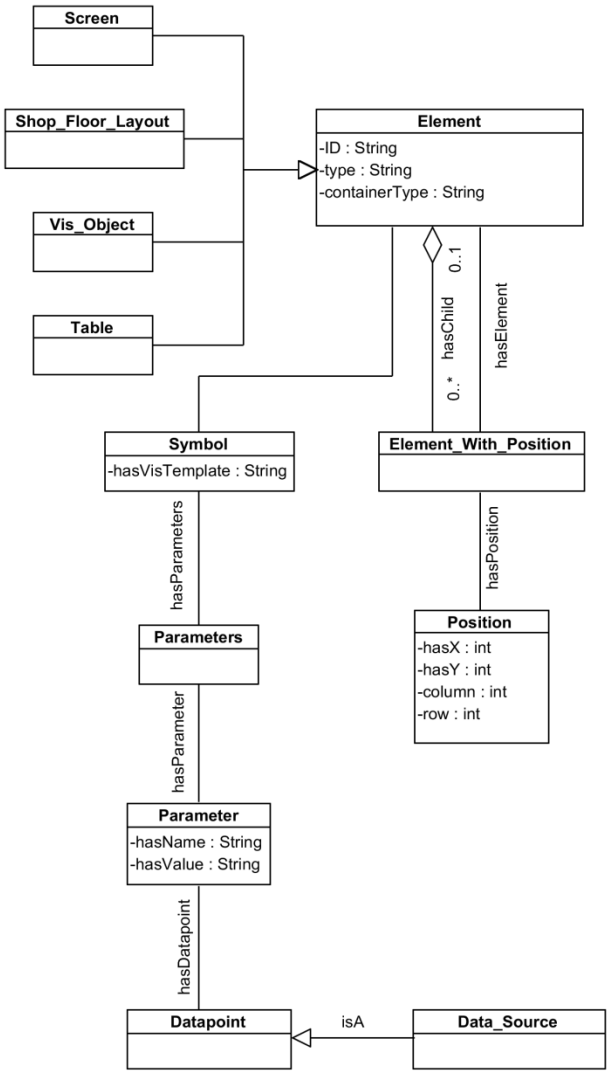


Figure 22. *Class diagram for representation of Visualization Information*

Thus, the different portions of the ontology which was designed on each step based on the perspective of physical layer, orchestrator layer and visualization layer were presented.

4.2 Use case Implementation

This section provides the use case definition followed by the description about the system; physical layer (section 4.2.3), orchestration layer (section 4.2.4) and visualization layer (section 4.2.5). The sections 4.2.3, 4.2.4 and 4.2.5 models the interactions needed between the layers and ontology which is step 6 of the proposed methodology. Then following step 7 of the methodology, ontology is instantiated based on system requirements captured in step 6. This is provided in section 4.2.6. Likewise section 4.2.7 provides the query templates required for interaction between the layers and ontology following step 8 of methodology. Thus the steps 6 to 8 of proposed methodology are provided in this section.

Finally the ontology services which are required for different layers to interact with ontology are provided in section 4.2.8.

4.2.1 Use case definition

For proving the functionality of the approach proposed in this thesis the ontology model designed will be tested on a manufacturing system. Based on the specific requirements of the system, the ontology will be extended and instantiated. The tools implementing the concepts of the approach will be developed and tested.

For this purpose, FASTory line installed in the Factory Automation and Systems and Technologies (FAST) Laboratory of TUT has been used as the test bed. The description of the system is provided in section 4.3. The implementation scenario is based on this system, FASTory Simulator, eScop orchestration tool and the ontology developed in this thesis.

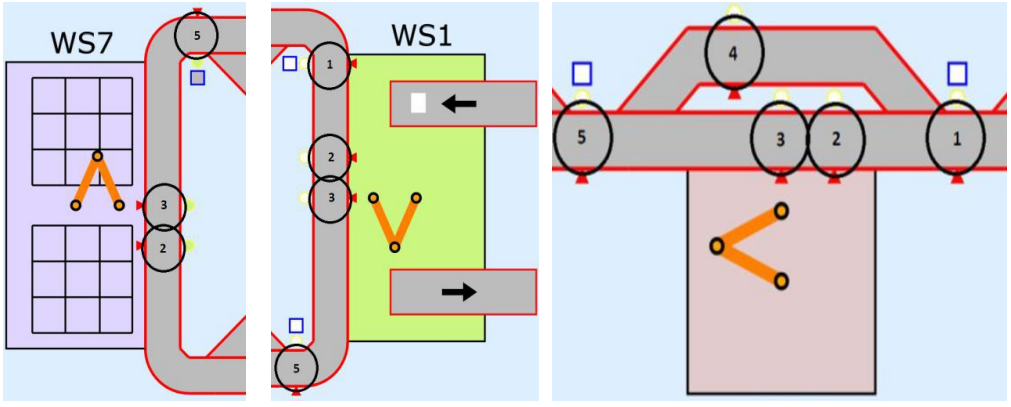
4.2.2 FASTory Line

The FASTory (Factory Automation Systems and Technologies Laboratory) production line is the system used as the testbed for implementation. The FASTory line demonstrates the assembly process of mobile phones. The photograph of the FASTory line is presented in figure 23. The line emulates the real operations of assembly process by drawing the phone components on pallets.

There are 12 work stations in the FASTory line. Work stations WS2-6 and WS8-12 are identical with a conveyor and a drawing robot. These workstations are for drawing the phone components on paper. WS1 is for loading papers on the pallets or for unloading papers. WS7 is a buffer station for loading or unloading and storing the pallets.

Figure 23. *FASTory Line*

The drawing workstations have 5 zones with a presence sensor to detect the pallets and a stopper to stop the pallets in each zone. Zone 1 is the entry point of the workstation and it has an RFID reader for pallet recognition. Zone 2 acts as an internal buffer for the workstation. The drawing process is performed in Zone 3. Zone 4 acts as the bypass for the conveyor to move the pallets to next workstation if Zone 3 is busy with drawing. Zone 5 is the exit point for the workstation.



a) b) c)

Figure 24. *a) Zones for WS7, b) Zones for WS, c) Zones for WS2-6, 8-12 [68]*

The drawing is performed for three main components of the mobile phone such as *frame*, *screen* and *keyboard.* Each component can be drawn with 3 different colors such as red, green or blue color and in 3 different shapes. The shape variations are shown in figure 25.

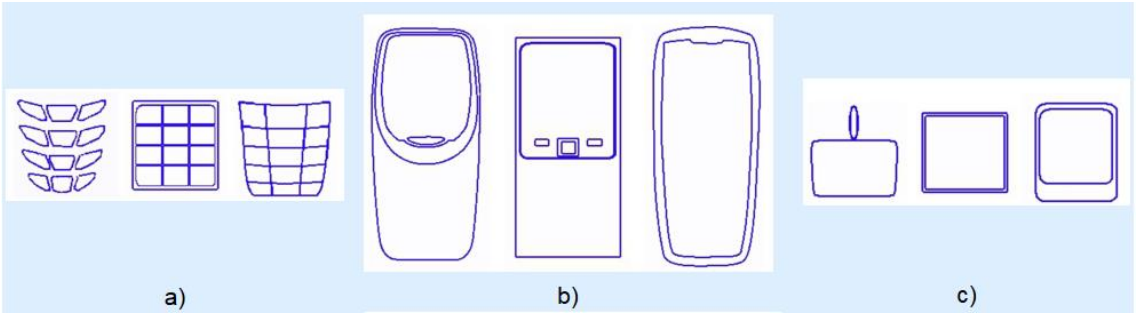


Figure 25. *a) Keyboard variations, b) Frame variations, c) Screen variations [68]*

The production process starts in WS7 where the pallet will be loaded. Then the pallet is moved to WS1 for loading the paper. It is then move through drawing workstations to draw the mobile parts. Finally, when all parts are drawn, the pallet again goes to WS1 to unload paper and then goes to WS7 to unload pallet or loads new paper in WS1 for another product.

The industrial controller Inico S1000 is used to connect to the sensors and actuators in shop-floor. It exposes the functionality of the devices as web services. By invoking the services, the production process can be controlled and monitored. The web services are deployed as RESTful web service. Currently, the RESTful services are still under development process. Hence to test the approach put forward in the thesis, a simulator which works the same as the FASTory line with third party applications is used.

4.2.3 FASTory Simulator

FASTory Simulator is the simulator developed for FASTory assembly line. It was developed by the researchers of eScop project to test their tools and being used for research, developments and education in TUT. The simulator provides robust results and makes it easy for users to test their tools as physical problems like power shut down and mechanical issues can be avoided with simulator [68]. It also provides visualization of the assembly process. Moreover, FASTory simulator is developed as a web application so it can be used on different operating systems and platforms. It uses the concepts of real line and works same as that of the real line when working with third party applications. For these reasons FASTory simulator is used in this thesis to support the implementation.

The FASTory simulator is available online [68].The server of the simulator is developed using node.js. The User Interface of simulator is built using HTML, CSS, SVG and javascript. It provides RESTful services to control the production process. Hence third party clients can use the RESTful service to invoke operation and monitor the production process. The structure of the simulator is shown in figure 26.

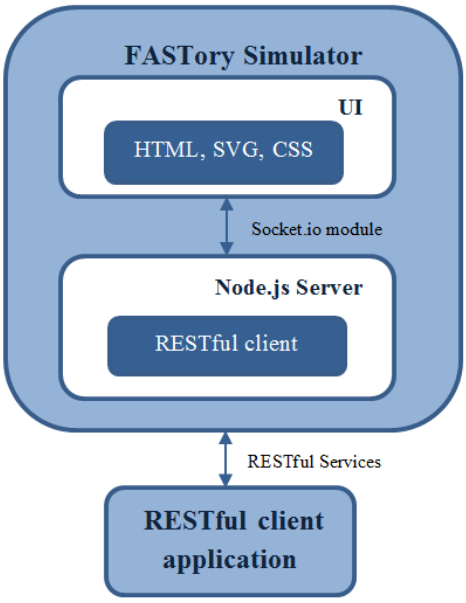


Figure 26. *FASTory Simulator architecture*

The RESTful services for invoking operations on line are provided using POST methods. The clients can also be notified of events occurring in line by subscribing to events. This is provided by using POST method to required event URL. Moreover, the data i.e. IO values can be obtained by using GET method. The detailed description of the web services provided for invoking operations, subscribing to events and getting IO values are provided in [68].

The step 6 of the proposed methodology is to capture the requirements of the system and model the interaction needed between PHL (FASTory simulator) and ontology. The requirement of the system from PHL perspective is to realize system configuration management using the ontology-driven approach. For this purpose, the device can use POST service to insert the device information to ontology whenever the device is started. In this way, the device information is stored in ontology at run-time. Whenever a new device is added to system, the information about the new device will also be stored in ontology upon startup of the device. Thus the configuration of the system can be managed dynamically.

FASTory simulator is implemented to send POST request to ontology services for registering the device on start up. The deviceRegistrationRequest contains description of the device information such as the id, type, topology and services offered by the device. This information can be inserted to MSO by using the ontology service. The sequence diagram for device registration is shown in figure 27.

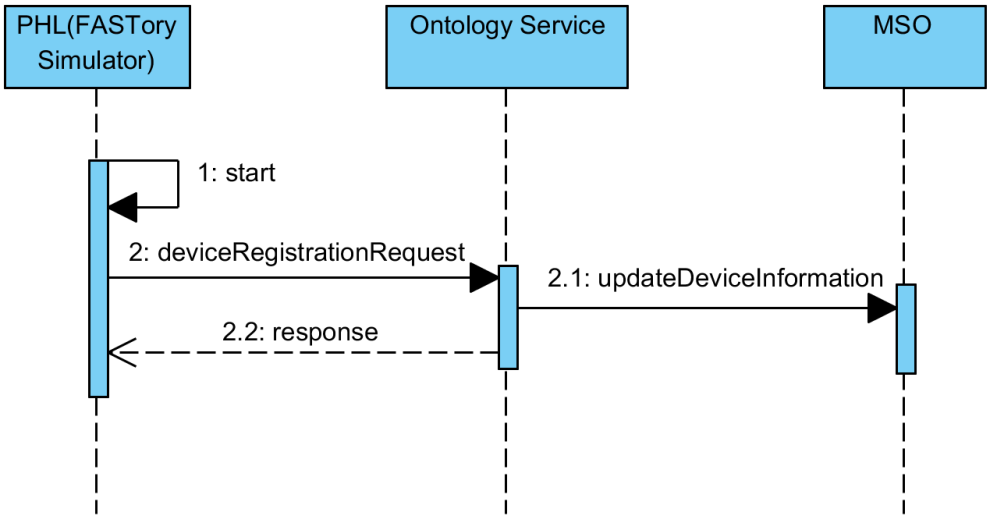


Figure 27. *Sequence diagram for device registration*

The ontology developed also manages the enterprise information such as orders. The information about orders is necessary for orchestrator to orchestrate the processes. The orders are input to the system at run-time. For this, FASTory simulator provides a user interface where the orders can be input by the user. The order information such as order id, customer name, customer address, product id and quantity are given by the user. The recipe information such as type of mobile component and color can also be entered in the interface. The screenshot of the web page is shown is figure 28.

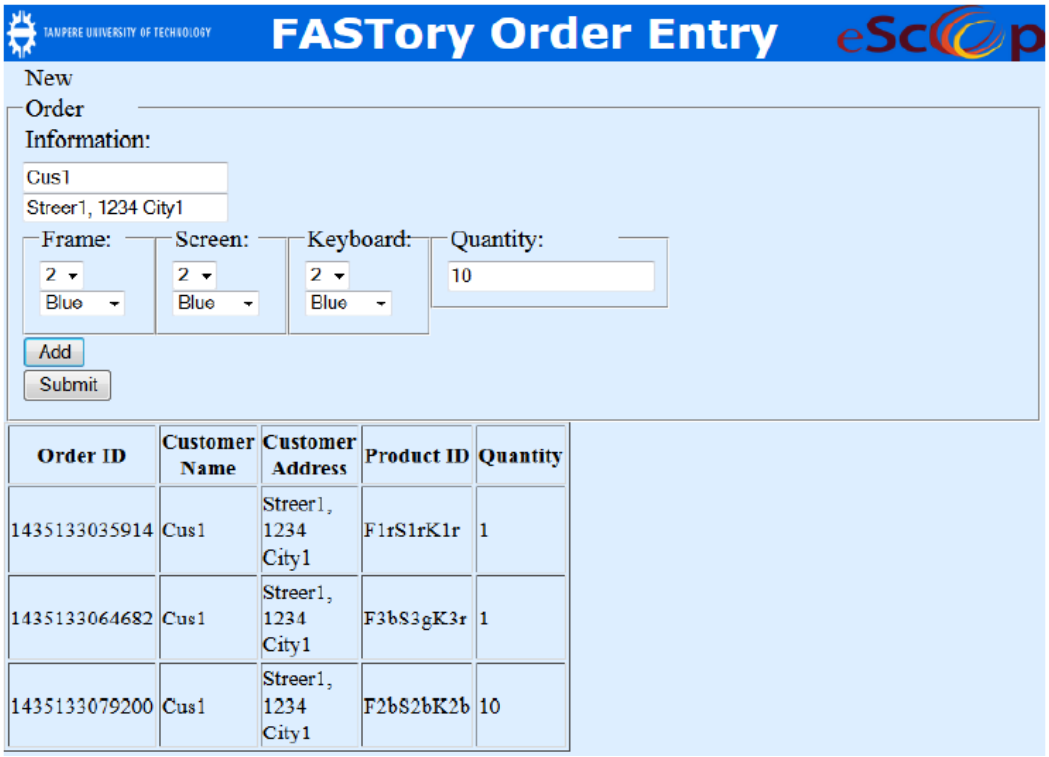


Figure 28. *Screenshot of FASTory Order Entry web page [68]*

When the order information is entered and submitted, the FASTory simulator sends a POST request (http://{host}:{port}/scheduling/order) to scheduling function with the order information. The scheduling function is developed separately as a tool in escop as one of the MES functions which provides the decision logic for orchestrator to invoke operations for scheduling. The scheduling function inserts the order in ontology whenever new orders are input in the Order entry web page. For this the ontology service can be used to insert the orders in MSO. The sequence diagram for the interaction between layers for order entry is shown in figure 29.

4.2.4 Orchestrator Layer

The orchestrator tool used in the implementation is also developed by researchers of eScop project. It controls the process execution. The orchestrator interacts with RPL and PHL. It interacts with RPL for requesting the knowledge about the system. It delegates the decision logic with help of MES functions and invokes services on PHL to control the process execution.

A particular case of production unit scheduling and dispatching in FASTory line is considered for the implementation of the thesis work. This demonstrates the support of ontologies for orchestration process. It is assumed that the pallets in the system are controlled by scheduling and dispatching functions. Whenever orders are input to the system, the order information is stored to ontology and the scheduling process modifies the PHL (i.e. changes pen color of robot) according to the ratio of each color needed in the registered orders. Then when a new pallet is detected, the scheduling function assigns the pallet to the order. After the robot finished drawing, it triggers DrawEndExecution event. Once the event is triggered, the scheduling function updates the order. This process requires knowledge representation of order, recipe, product and pallet. With ontologies new orders can be inserted and orders can be updated when new pallets are assigned to orders or when a process defined by recipe is finished. The sequence diagram for scheduling scenario is shown in figure 29.

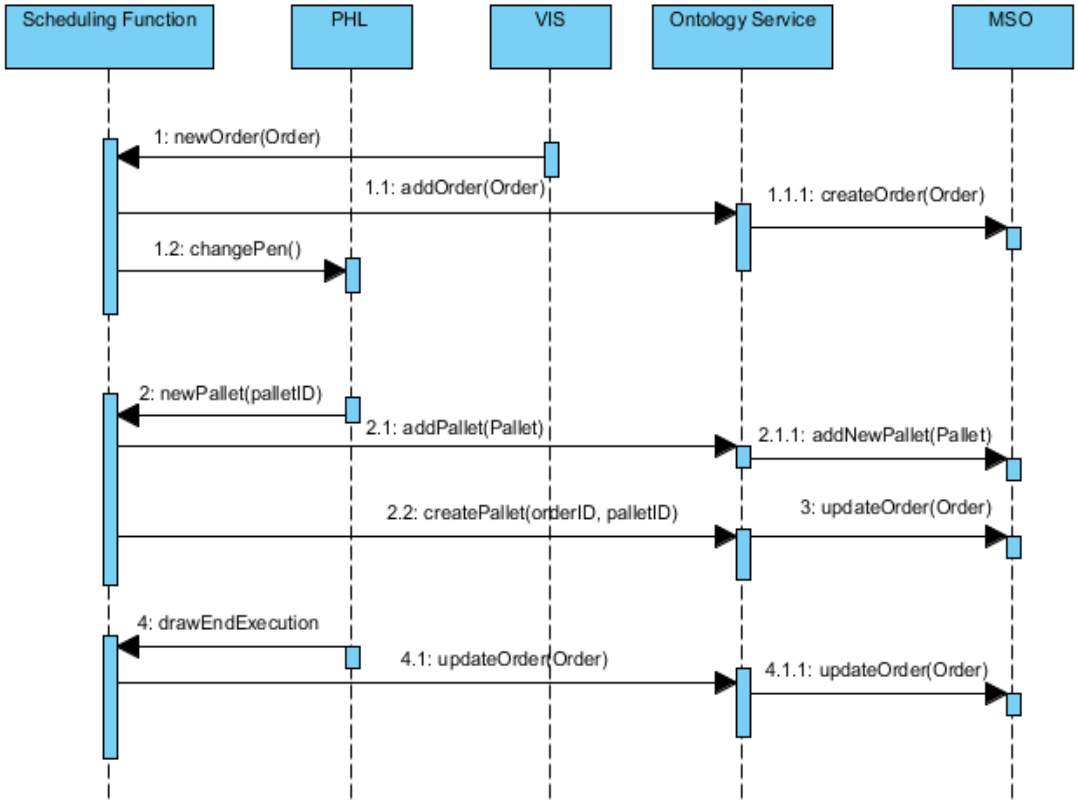


Figure 29. *Sequence diagram for scheduling scenario in FASTory Line*

For performing dispatching function, a decision algorithm is needed for zone 1 to decide whether the pallet needs service from the particular workstation. If the services required for the product are not provided in the workstation then it is moved from the workstation in predefined optimal path. In case the services provided in the workstation are required for the product then the algorithm checks if the route to the required zone is free or not. If it is not free then again it is moved from workstation in predefined path. Otherwise it is moved to the required zone where the operations are executed on the product. The flow chart of the decision algorithm is shown in figure 30. The decision algorithm requires the representation of the required knowledge in ontology.

The decision in zone 1 is based on whether the pallet needs service from the particular workstation. For this, the description of services provided by the robot in particular workstation and pallet recipe needs to be compared to make the decision. In FASTory case, all the technical descriptions for invocation of the operations are embedded in the URL of the service. Hence the list of service URLs for a particular robot and the pallet recipe can be queried from ontology. Based on it a list of executable URLs which invokes real operations on the system is retrieved and ORL orchestrates these services to serve the pallet. The sequence diagram for the dispatching scenario is shown in figure 31.

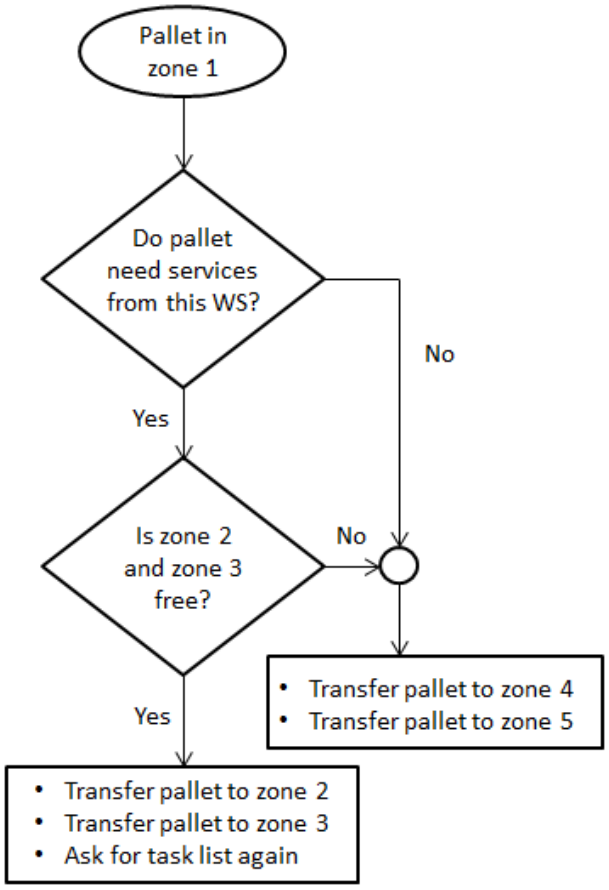


Figure 30. *Flow chart of decision logic algorithm for zone 1*

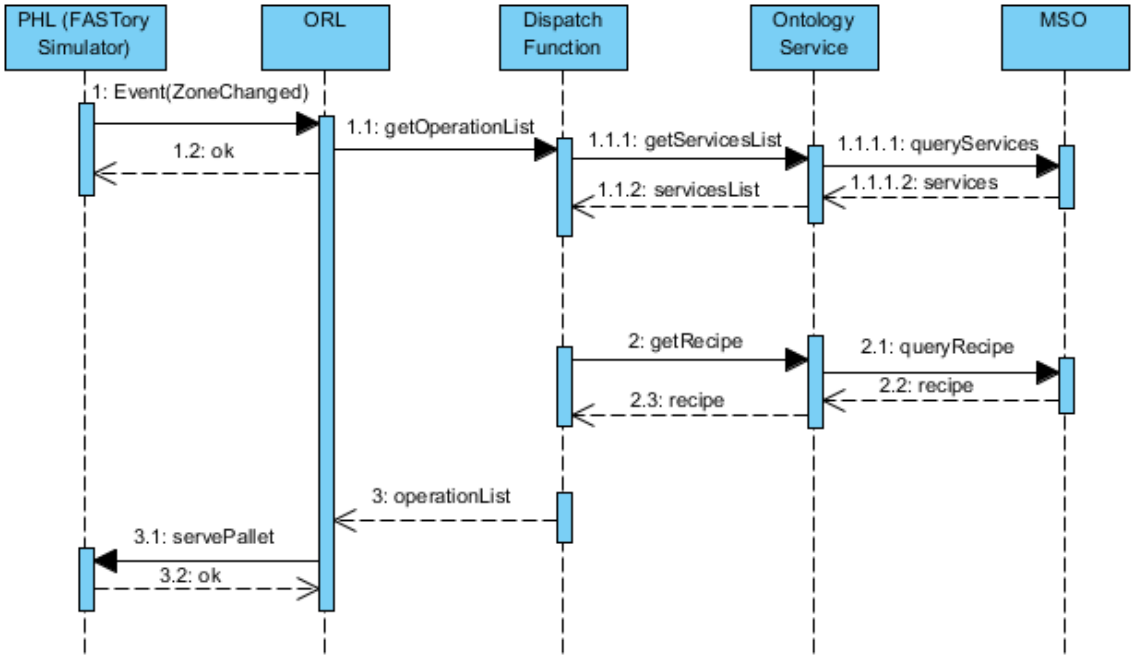


Figure 31. *Sequence diagram for dispatching scenario in FASTory Line*

4.2.5 Visualization Layer

The visualization layer is the front-end of the system, which provides visualization of the manufacturing system in a web browser. The visualization is built as a web application which provides visualization of the factory-shop floor layout based on the system topology, graphical details of screen elements and screen configuration. To test the dynamic re-configuration capability of visualization, the application is built such that it requests the visualization information in ontology to generate the visualization layout. If there are any changes to system at run-time like addition of new device on shop-floor, the ontology will be updated with device information and re-uses its description to generate visualization information for the new device. Visualization application can be reconfigured by simply requesting the visualization information for the screen and its graphical elements again from ontology and generating the layout.

The visualization application is built using technologies like HTML5, CSS3 and JavaScript (Angular JS and D3.js). AngularJS is a framework for developing client-side dynamic web applications. It provides automatic synchronization of data from UI with JavaScript objects through two-way data binding. It also allows sending asynchronous JavaScript and XML (AJAX) requests to remote servers. AJAX is a group of web technologies (HTML, JS, XML etc.,) that allows the application to communicate to server in the background without interfering with the display of existing web page. The application uses AJAX to request information about graphical elements from ontology to generate visualization. VIS provides two screens; configuration screen and SVG visualization screen.

The metadata such as position of the visual element may change from one screen to another. Hence this metadata needs to be provided by user at the time of configuration of the visualization screen. Then the positions can be stored in ontology and used for displaying the elements in screen. The positions of the visual elements can also be calculated automatically for e.g. in case of shop-floor elements. If the topology of the system components is known, then algorithms can be employed which calculates the position of the elements based on topology and size of symbols. The employment of algorithms to calculate positions may also need some data from users e.g. the starting point for system from which the layout will be generated. Moreover, there may also be more than one symbol template for the same visual element. The different templates are needed because different screens may require the visualization elements with different symbols. In such case, the choice of the templates for the particular screen should also be provided by user during screen configuration. Thus users provide the information needed for configuration of screen. For this purpose, VIS provides a configuration screen where the required metadata can be collected from user and then provides the SVG visualization.

The configuration screen provided includes a graph layout of the shop-floor. VIS uses Data Driven Documents (D3.js) which is a JavaScript library for producing dynamic visualizations in web- browser. It uses SVG, HTML5 and CSS standards. VIS requires the information needed to create the graphs to be in GraphJSON format. GraphJSON format includes definition of nodes and links. The graphJSON example for screen with shop-floor layout consisting of three workstations WS1, WS2 and WS3 is shown below.

```
{
"nodes":[{ "id":0,
            "caption":"WS1",
            "metadata":{"x":0,"y":0},
           },
           { "id":1,
             "caption":"WS2",
             "metadata":{"x":0,"y":0},
            },
            { "id":2,
              "caption":"WS3",
              "metadata":{"x":0,"y":0},
           }],
 "links":  [
          {"source":0,"target":2,"caption":"isbefore"},
          {"source":2,"target":1,"caption":"isbefore"}
         ]
}
```

D3.js supports this format of data to ease the generation of graphs. Hence RPL should provide VIS with the information about the visualization screen in graphJSON format. Here nodes represent the system components and links define the relationship between the components. The configuration screen generates the layout automatically by employing algorithm which determines the position of the nodes by using the relationships defined by links. It also provides options for users to drag drop the nodes to define the position of the graphical elements on screen or enter the positions (x, y values) in a table. By pressing the SAVE LAYOUT button, the position information is sent to RPL to be saved in ontology. The configuration screen is shown is figure 32.

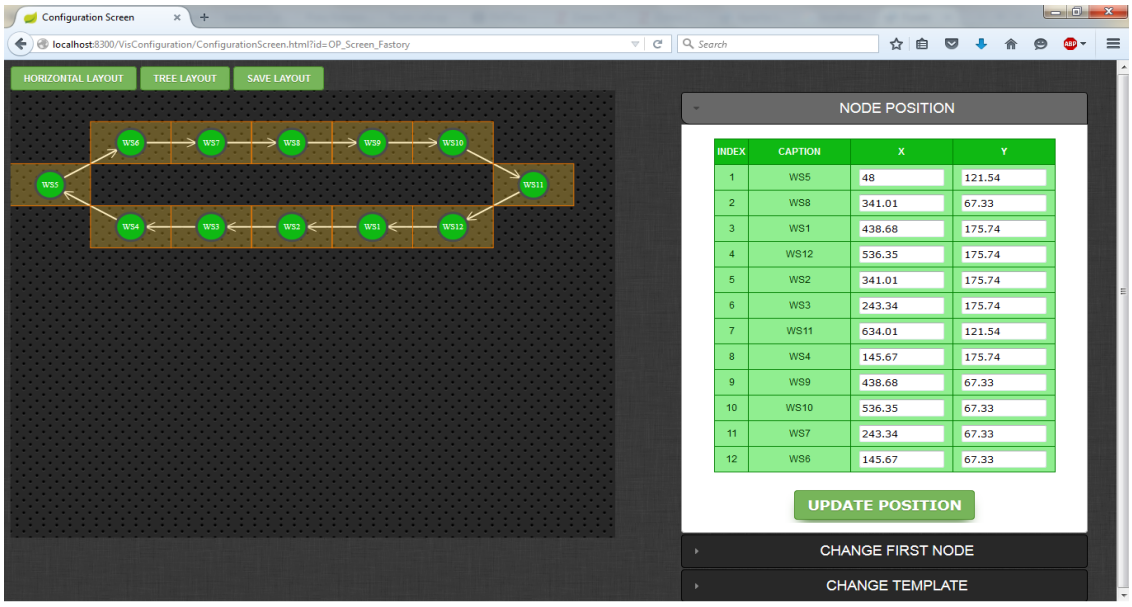


Figure 32. *Configuration screen for visualization*

The SVG visualization screen is provided by using the information about graphical elements such as visualization screen composition (e.g. shop-floor layout, tables, etc..,) and position of the graphical element on screen. The ontology can represent the graphical elements and provide information regarding the id, type, visualization template, container type, symbols, visualization parameters and child elements for each graphical element. This information is needed by VIS for creating the graphical elements in screen. VIS also stores various symbol templates defined using SVG. It chooses the symbol template based on the reference provided by the *visualization template* information from ontology. JSON format is selected as a data format for receiving the visualization information about elements from ontology since it is a lightweight data-interchange format. Ontology service can query the required information and sent to VIS in JSON format. Example JSON format for shop-floor layout element is shown below.

```
{
"containerType": "Canvas",
"id": "Shop_Floor_Layout",
"visTemplate": "Vis_Shop_Floor_Layout",
"type": "Vis_Shop_Floor_Layout",
"children":
      {
        "http://localhost:8300/RPL/VIS/WS1": {"x": 0.6110085, "y": 0.43764144},
        "http://localhost:8300/RPL/VIS/WS2": {"x": 0.47366476, "y": 0.09974026},
      }
}
```

The interaction needed between VIS and RPL (ontology services and MSO) is shown in figure 33.

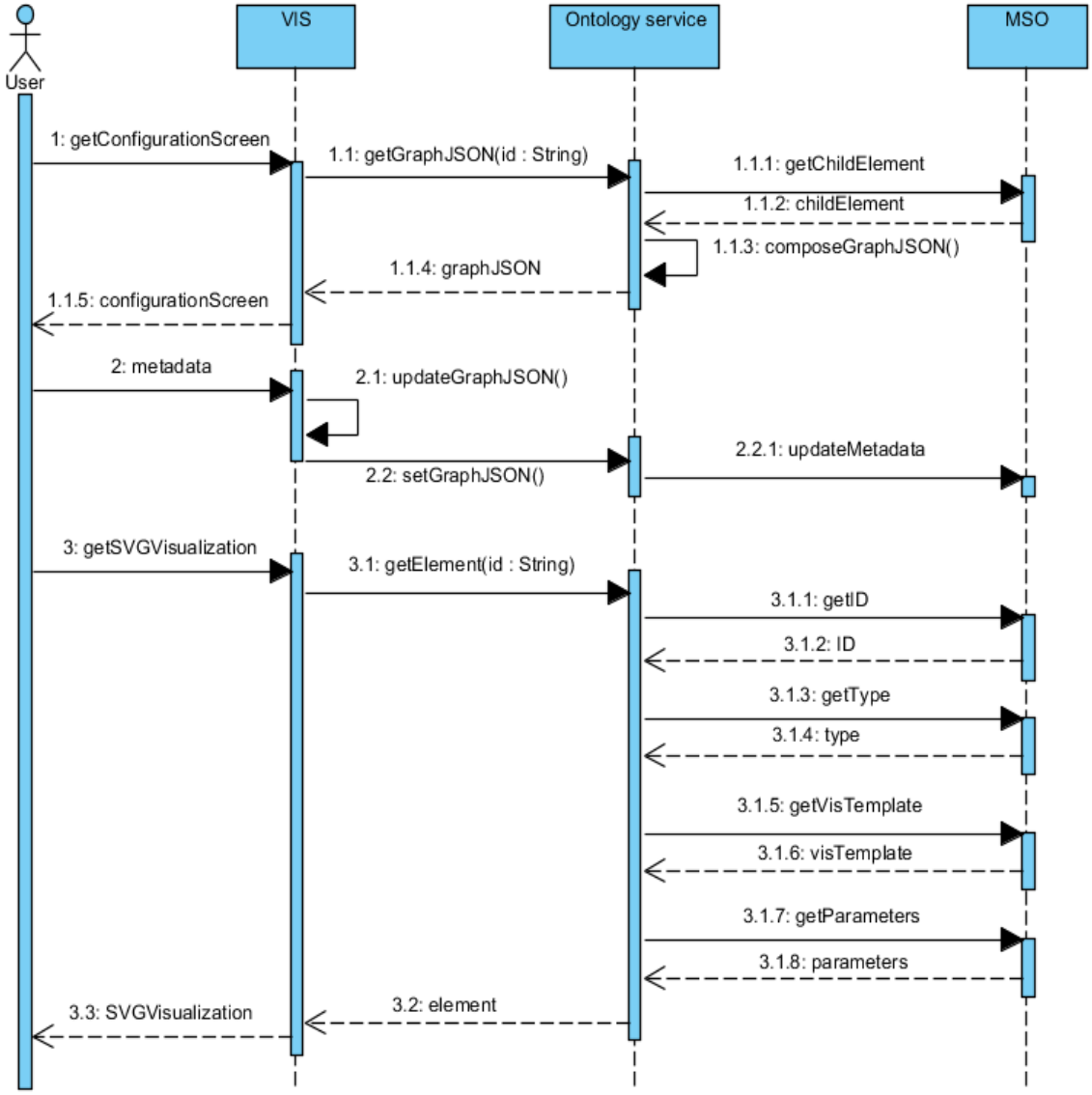


Figure 33. *Sequence diagram for interaction between VIS and RPL*

4.2.6 Instantiation of MSO

The ontology developed is instantiated for FASTory system. The interactions needed between the layers and ontology is presented in sections 4.2.3, 4.2.4 and 4.2.5. Based on these requirements instantiation is carried out. The components in the FASTory system that needs to be represented in ontology are workstations, conveyors, robots and pallets. The components can be represented in ontology under *Component* class which was discussed in section 4.1.1. Workstations can be defined under *Subsystem* class. Likewise, conveyors under *Conveyor* class, robots under *Equipment* class and pallets under *Container* class. The services offered by the conveyors and robots are defined under *Service* class. The instantiation of these components are not performed initially but are instanti-

ated at run-time to demonstrate the capability of ontology to maintain the configurations at run-time.

The FASTory system takes orders for drawing phone components on paper. Each order includes information about the type of product, quantity needed and recipe. The orders are entered to the system from UI and it can be stored in ontology by creating the instances to represent the information. It is done with the classes and properties discussed in section 4.1.3. In FASTory line, the recipe is the specifications for the mobile parts. Each part should be drawn in a particular shape and color. Hence the recipe should define the part, shape and color. These specifications are represented by creating datatype properties; *form*, *shape* and *color* for *Recipe_Row* class. The new class diagram for order and recipe representation with these properties is shown in figure 34. With this representation each recipe row defines the color and shape for a particular part of the phone. Thus this portion of ontology is extended to suit FASTory system.

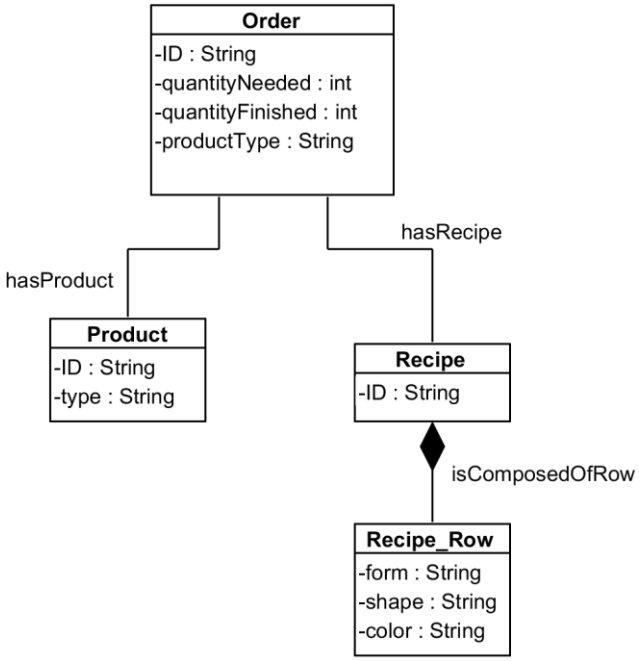


Figure 34. *Class diagram for order and recipe representation with extension to Recipe_Row class*

For storing the visualization information in ontology, the representation discussed in 4.1.4 is sufficient. The screens for visualization can contain any graphical element like Shop-floor layout, table, etc. and the screen composition varies from screen to screen. Hence the screen composition is defined in ontology initially. For the implementation the screen is considered to be composed of only the shop-floor layout. The instantiation in ontology is shown in figure 35. It is seen that *FASTory_Screen* instance is created as a screen element. The id and container type are specified. The *FASTory_Screen* has child element *Shop_Floor_Layout_With_Position*. This element is instantiated in *Element_With_Position* class. At the time of instantiation the position of the layout in the

screen is not known. Hence position is not defined yet. The instance is mapped back to the *shop_Floor_Layout* element which is instantiated in *Shop_Floor_Layout class.*

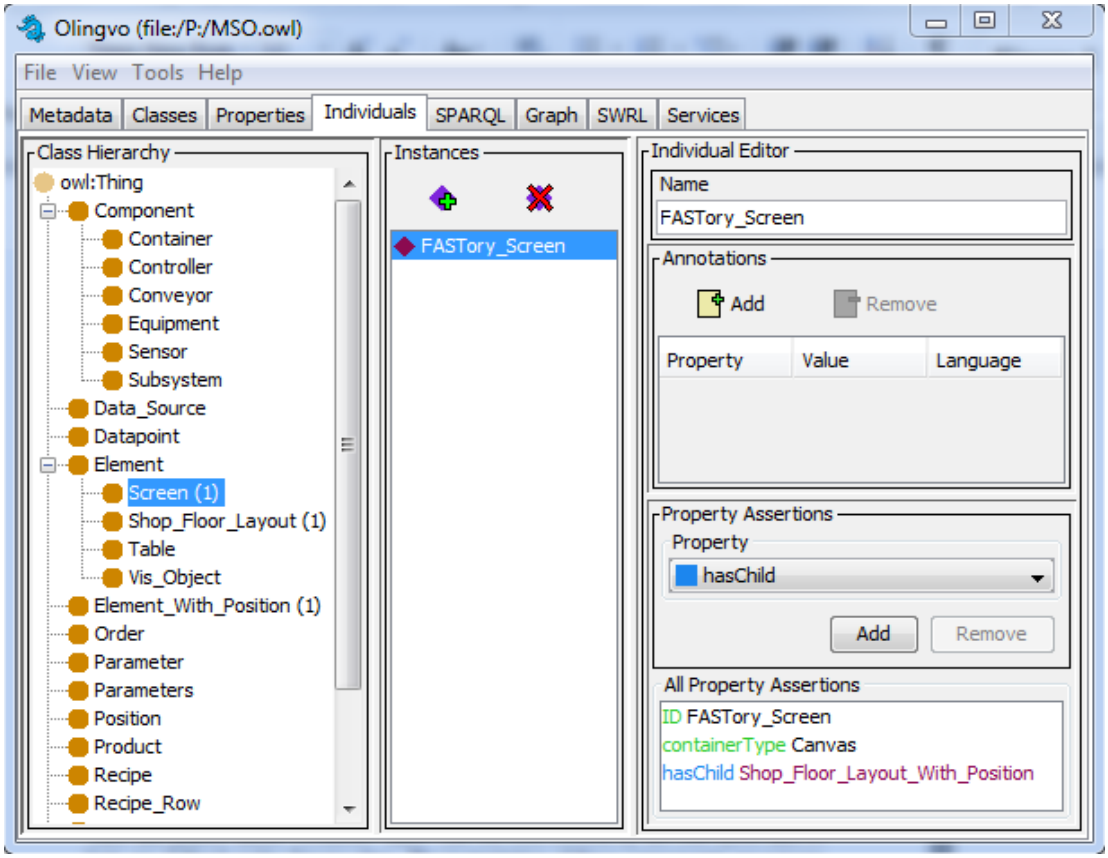


Figure 35. *FASTory screen instantiation*

The *shop_Floor_Layout* in turn includes the graphical elements of workstations as its child elements. The thesis demonstrates a dynamic re-configurable visualization by considering the conditions when a new workstation is added to system or when a workstation in the system becomes inactive. Hence the child elements of the *shop_Floor_Layout* (i.e. graphical element representing work stations) can change at run-time depending on the devices that are active in the system. So the instantiations of the graphical elements for workstations are performed at run-time.

It should also be noted that the graphical elements representing workstations can also have child elements (e.g. graphical elements representing conveyors, robot etc.). In this way the levels of representation can be increased based on which hierarchical levels the system needs to be visualized. For the implementation carried out in this thesis, workstations are considered as the lowest level in the hierarchy for visualization.

4.2.7 Query templates

The query templates are developed based on the interaction needed between different layers and MSO which were presented in section 4.2.3, 4.2.4 and 4.2.5. The interactions

modeled between **Ontology service** and **MSO** in sequence diagrams provided in figure 27, 29, 31 and 33 are the required queries. The template for these queries is shown in table 2. The words marked in red colour can be replaced accordingly while implementing the templates. The prefix to be used along with the template is,

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX mso:<http://www.escop-project.eu/MSO.owl#>
```

Table 2. Query templates

Query name and description	**Query template**

insertDeviceInformation
 	
Query to insert device information to MSO
	INSERT {
	?comp rdf:type mso:<class>.
	?comp mso:ID "<id>"^^xsd:string.
	?comp mso:type "<component type>"^^xsd:string.
	?comp mso:isAfter mso:<before component>.
	?comp mso:isBefore mso:<after component>.
	?comp mso:includes mso:<components[i]>.(repeat for each value i of *components*)
	?comp mso:belongsTo mso:<parent component>.
	?comp mso:hasServiceList mso:list_<id>.
	?list_<id> rdf:type mso:Service_List.
	?list_<id> mso:hasService mso:service_<i>.
	?service_<i> rdf:type mso:Service.
	?service_<i> mso:hasServiceOperation ?operation_<i>.
	?operation_<i> rdf:type Service_Operation.
	?operation_<i> mso:hasLink "<services[i]>"^^xsd:string. (repeat for each value i of *services*)
	}
	WHERE{
	BIND(IRI("mso:<id>")) AS ?comp)
	}

deleteDeviceInformation
 	
Query to delete device information in MSO.
	DELETE {
	?comp rdf:type ?class.
	?comp mso:ID ?id.
	?comp mso:type ?type.
	?comp mso:isAfter ?isAfter
	?comp mso:isBefore ?isBefore.
	?comp mso:includes ?components.
	?comp mso:belongsTo ?parent.
	?comp mso:hasServiceList ?list.
	?list rdf:type mso:Service_List.
	?list mso:hasService ?service.
	?service rdf:type mso:Service.
	?service mso:hasServiceOperation ?operation.
	?operation rdf:type Service_Operation.
	?operation mso:hasLink ?link.
	}
	WHERE{

	?comp mso:ID "<component id>"^^xsd:string. }

getDeviceInformation
 	
Query to get device information from MSO
	SELECT ?id ?type ?cls ?components ?isBefore ?isAfter ?belongsToComponent ?services
	WHERE{
	?comp rdf:type ?cls.
	?comp mso:ID ?id.
	?comp mso:type ?type.
	?comp mso:isAfter ?isAfter
	?comp mso:isBefore ?isBefore.
	?comp mso:includes ?components.
	?comp mso:belongsTo ?belongsToComponent.
	?comp mso:hasServiceList ?list.
	?list mso:hasService ?service.
	?service mso:hasServiceOperation ?operation.
	?operation mso:hasLink ?services.
	?comp mso:ID "<component id>"^^xsd:string.
	}

getDevices
 	
Query to get IDs of all devices
	SELECT ?id
	WHERE{
	?comp mso:ID ?id.
	{?comp rdf:type mso:Subsystem.}OPTIONAL
	{?comp rdf:type mso:Conveyor.}OPTIONAL
	{?comp rdf:type mso:Equipment.}
	}

getOrders
 	
Query to get all orders from MSO
	SELECT ?id
	WHERE {
	?order a mso:Order.
	?order mso:ID ?id.
	}

deleteOrders
 	
Query to delete all orders
	DELETE{
	?order rdf:type mso:Order.
	?order mso:ID ?id.
	?order mso:hasRecipe ?res.
	?res rdf:type mso:Recipe.
	?res mso:ID ?resId.
	?res mso:isComposedOfRow ?res_row.
	?res_row rdf:type mso:Recipe_Row.
	?res_row mso:component ?form.
	?res_row mso:type ?shape.
	?res_row mso:color ?color.
	?order mso:isComposedOfRow ?row.
	?row rdf:type mso:Order_Row.
	?row mso:quantity ?q.
	?row mso:done ?d.
	?prd mso:belongsToOrder ?order.
	}WHERE{
	{?order a mso:Order.}OPTIONAL
	{?order mso:hasRecipe ?res.
	?res mso:ID ?resId.
	?res mso:isComposedOfRow ?res_row.
	?res_row mso:component ?form.
	?res_row mso:type ?shape.
	?res_row mso:color ?color.}OPTIONAL
	{?order mso:isComposedOfRow ?row.
	?row mso:quantity ?q.
	?row mso:done ?d.}OPTIONAL

	{?prd mso:belongsToOrder ?order.} }

getOrder
 	
Query to get order
	SELECT ?id ?quantity ?done \n"
	WHERE {
	?order mso:ID <**order id**>
	?order mso:ID ?id.
	?order mso:isComposedOfRow ?row.
	?row mso:quantity ?quantity.
	?row mso:done ?done.
	}

getOrderRecipe
 	
Query to get recipe of order
	SELECT ?id ?form ?shape ?colour
	WHERE {
	?order mso:ID “<**id**>”^^xsd:string.
	?order mso:hasRecipe ?recipe.
	?recipe mso:ID ?id.
	?recipe mso:isComposedOfRow ?rows.
	?rows mso:component ?form.
	?rows mso:type ?shape.
	?rows mso:color ?colour.
	}

getOrderPallet
 	
Query to get pallet from order
	SELECT ?id
	WHERE {
	?order mso:ID “<**id**>”^^xsd:string.
	?pallet mso:belongsToOrder ?order.
	?pallet mso:ID ?id.
	}

insertOrder
 	
Query to insert order
	PREFIX fn:<http://www.w3.org/2005/xpath-functions#>
	INSERT {
	?order rdf:type mso:Order.
	?order mso:ID "<id>"^^xsd:string.
	?order mso:hasRecipe ?recipe.
	?order mso:isComposedOfRow ?row.
	?row rdf:type mso:Order_Row.
	?row mso:quantity <quantity of product needed>^^xsd:int.
	?row mso:done <quantity of product finished>^^xsd:int.
	}
	WHERE{
	BIND(IRI(fn:concat("mso:order_","<**order id**>") AS ?order)
	BIND(IRI(fn:concat("mso:order_row_","<**order id**>") AS ?row)
	?recipe mso:ID <**recipe id**>^^xsd:string.
	}

deleteOrder Query to delete order	DELETE{ ?order rdf:type mso:Order. ?order mso:ID <**order id**>^^xsd:string. ?order mso:hasRecipe ?res. ?res rdf:type mso:Recipe. ?res mso:ID ?resId. ?res mso:isComposedOfRow ?res_row. ?res_row rdf:type mso:Recipe_Row. ?res_row mso:component ?form. ?res_row mso:type ?shape. ?res_row mso:color ?color. ?o mso:isComposedOfRow ?row. ?row rdf:type mso:Order_Row. ?row mso:quantity ?q.

	?row mso:done ?d.
	?prd mso:belongsToOrder ?order.
	}
	WHERE{
	{?order mso:ID <order id>^^xsd:string.}OPTIONAL
	{?order mso:hasRecipe ?res.
	?res mso:ID ?resId.
	?res mso:isComposedOfRow ?res_row.
	?res_row mso:component ?form.
	?res_row mso:type ?shape.
	?res_row mso:color ?color.}OPTIONAL
	{?o mso:isComposedOfRow ?row.
	?row mso:quantity ?q.
	?row mso:done ?d.}OPTIONAL
	{?prd mso:belongsToOrder ?order}.
	}

getPallets
 	
Query to get IDs of all pallets
	SELECT ?id
	WHERE {
	?pallet a mso:Container.
	?pallet mso:ID ?id.
	}

getPallet
 	
Query to get pallet information
	SELECT ?id ?currentReciepe_id ?form ?shape ?colour ?parentOrder
	WHERE {
	{?pallet mso:ID <pallet id>^^xsd:string.
	?pallet mso:ID ?id.}OPTIONAL
	{?pallet mso:currentRecipe ?c_res.
	?c_res mso:ID ?currentReciepe_id.
	?c_res mso:isComposedOfRow ?c_rows.
	?c_rows mso:component ?form.
	?c_rows mso:type ?shape.
	?c_rows mso:color ?colour.}OPTIONAL
	{?pallet mso:parentOrder ?parentOrder.}
	}

insertPallet
 	
Query to insert pallet information
	INSERT {
	?p rdf:type mso:Container.
	?p mso:ID <pallet id>^^xsd:string.
	?p mso:currentRecipe ?c_res.
	?p mso:parentOrder <parent order id>^^xsd:string.
	WHERE {
	BIND(IRI(fn:concat("mso:pallet_",<pallet id>)) AS ?p)
	?c_res mso:ID <recipe id>^^xsd:string.
	}

getChildElement
 	
Query to get child elements of an element
	SELECT ?childElement
	WHERE{
	mso:<element> mso:hasChild ?elementWithPosition.
	?elementWithPosition mso:hasElement ?childElement.
	}

getMetadata Query to get metadata i.e. position of child element in parent ele-	SELECT ?x ?y WHERE{ mso:<parentElement> mso:hasChild ?element_with_position. ?element_with_position mso:hasElement mso:<childElement>. ?element_with_position mso:hasPosition ?position. ?position mso:hasX ?x.

ment	?position mso:hasY ?y. }

getNextElement
 	
Query to get the element physically located after an element.
	SELECT ?nextElement
	WHERE{
	?comp mso:hasGraphicalElement mso:<element>.
	?comp mso:isBefore ?nextComp.
	?nextComp mso:hasGraphicalElement ?nextElement.
	}

getElement
 	
Query to get ID of element
	SELECT ?id
	WHERE {
	mso:<element> mso:ID ?id.
	}

getVisTemplate
 	
Query to get visualization template of element
	SELECT ?template
	WHERE {
	mso:<element> mso:visTemplate ?template.
	}

getContainerType
 	
Query to get container type of element
	SELECT ?containerType
	WHERE {
	mso:<element> mso:containerType ?containerType.
	}

getType
 	
Query to get type of element
	SELECT ?type
	WHERE {
	mso:<element> mso:hasType ?type.
	}

getParameters
 	
Query to get symbol parameters of element
	SELECT ?parameterName ?parameterValue
	WHERE{
	mso:<element> mso:hasSymbol ?symbol.
	?symbol mso:hasParameters ?ps.
	?ps mso:hasParameter ?parameter.
	?parameter mso:hasName ?parameterName.
	?parameter mso:hasValue ?parameterValue.
	}

4.2.8 Implementation of ontology services

The ontology services are developed to expose the knowledge about the system in ontology to all layers in the system architecture. The services are offered for PHL, ORL and VIS in order to support device configuration management, orchestration of processes and re-configurable visualization respectively. The services created are based on the interactions needed between

- PHL (using FASTory simulator) and MSO
- ORL and MSO

- VIS and MSO

The ontology services are developed as RESTful web services in a JAVA programming environment using Netbeans IDE. RESTful web services are created using Spring framework. The services are implemented using 4 internal modules which interact with other layers. The internal modules are developed to group the services offered to each layer for easier handling of information to be stored or retrieved from MSO. The architecture for ontology services is shown in figure 36.

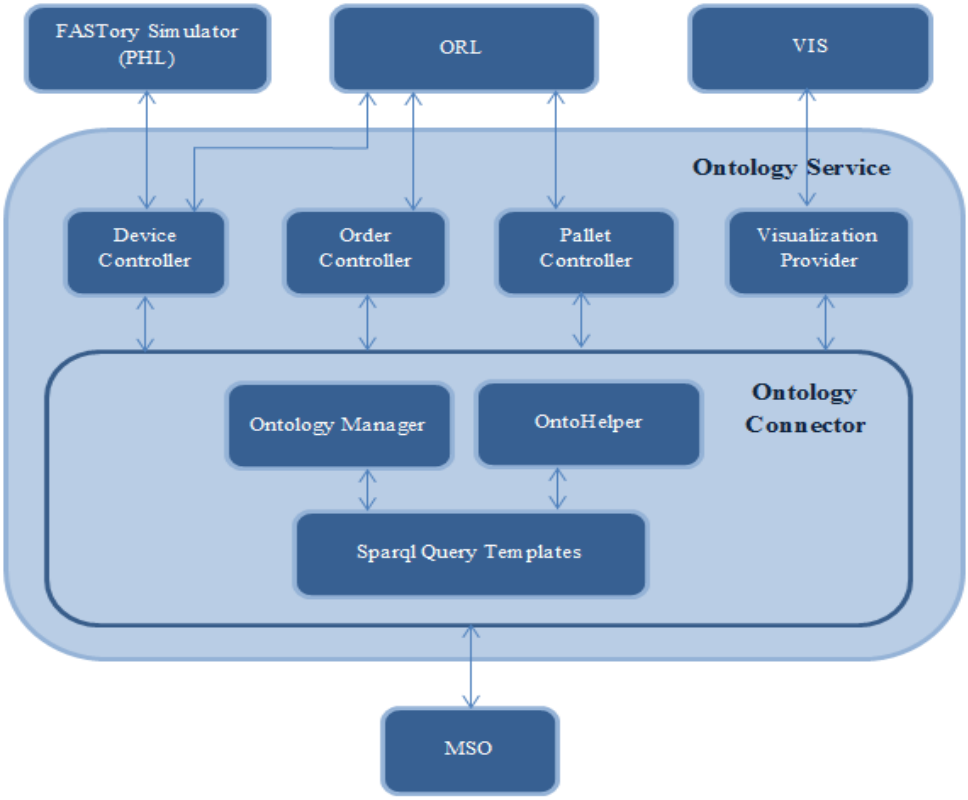


Figure 36. *Architecture of Ontology service*

The description of the modules are shown in table 3.

Table 3. *Description of service modules*

Module	**Description**
Device Controller	It offers services to register or unregister the devices in MSO. PHL uses this service to store the existence of the device and its capabilities in MSO. It also provides service to expose the device description and capabilities to ORL.

Order Controller	It offers service to insert, delete and update the orders in MSO. The services provided by this module are used by ORL during scheduling and dispatching operations.
Pallet Controller	It offers service to insert, delete and update pallets in MSO. These services are also used by ORL during scheduling and dispatching operations.
Visualization Provider	It offers services to expose or update the screen configuration information and services for exposing the visualization information of graphical elements. The services offered by this module are used by VIS layer.

4.2.8.1 Device Controller Module

The device controller module is developed to offer services mainly for PHL to interact or manipulate the device information in MSO. However, it also provides services for ORL to learn about device information for executing orchestration process. The interaction of DeviceController with PHL and ORL is shown in the sequence diagram in figure 37. The messages sent to DeviceController are the service requests from layers. The messages between DeviceController and MSO represent the queries used to manipulate the information in MSO.

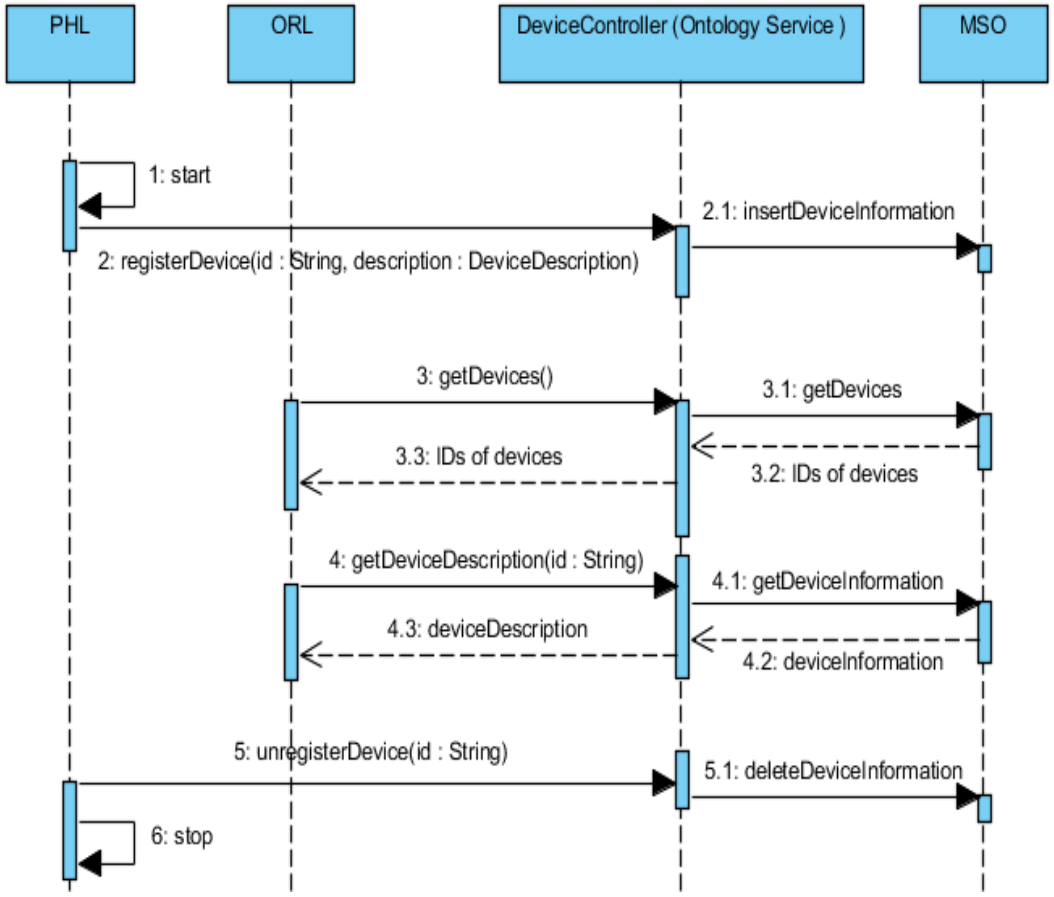


Figure 37. Sequence diagram for DeviceController interactions

DeviceController class is built as a REST controller in the Spring framework. Each method in the class defines a service route and it is triggered when servlet dispatcher receives the HTTP request to the routes defined in it. The DeviceController class depends on OntologyManager class to fulfill the service requests. It uses the query templates defined in SparqlQueryTemplate class to manipulate the information in MSO to fulfill the request. DeviceDescription class is a mapping class to the JSON message which the RTU sends to RPL. The message schema for device description is shown below,

{
  id(String) : Unique identifier for RTU
  type(String) : Type of device
  cls(String) : Category of device (e.g. robot, conveyor, etc.)
  components (List<String>) : List of Ids of the child components of the device
  isBefore(String) : Id of the device before which this device is present
  isAfter(String) : Id of the device after which this device is present
  belongsToComponent(String) : Id of the component to which the device belongs.
  services (List<String>) : List of service URLs
}

The class diagram of deviceController is shown in figure 38. The description of the methods in deviceController class is provided in table 4.

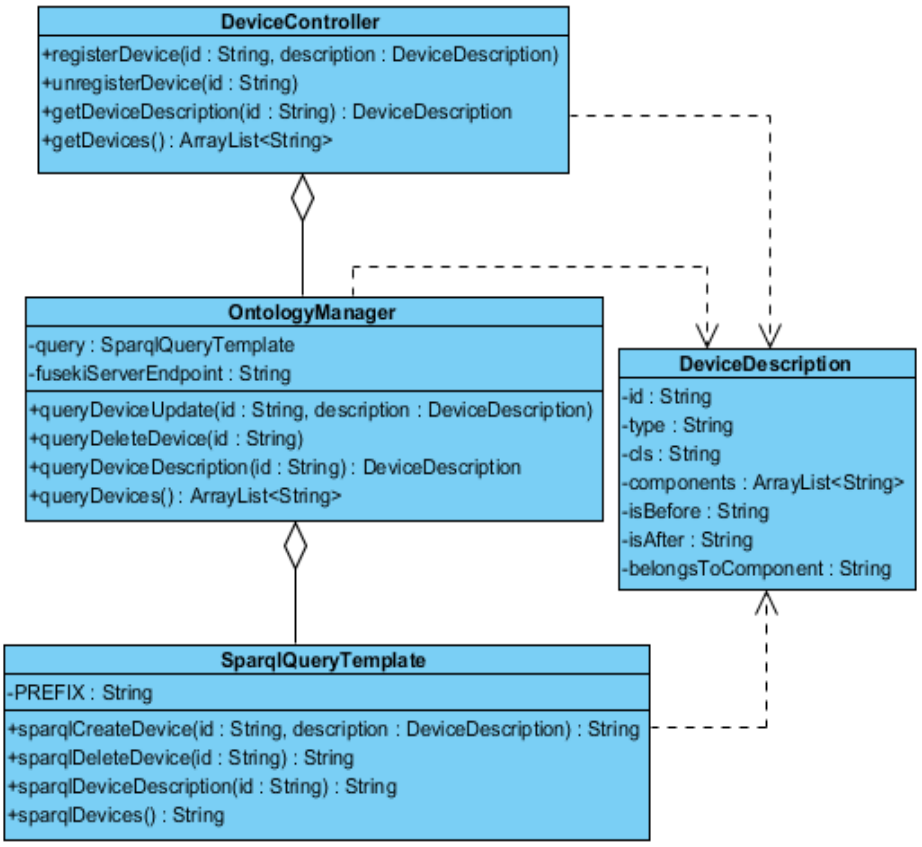


Figure 38. *Class diagram of DeviceController module*

Table 4. Description of methods in DeviceController class

Method	**registerDevice(id : String, description : DeviceDescription)**
Route	/RTU/{deviceID}
Request method	POST
Description	This method registers the device information to MSO. The service is used by PHL on initialization of devices.
Method	**unregisterDevice(id : String)**
Route	/RTU/{deviceID}
Request method	DELETE
Description	This method deletes the device information from MSO. The service is used by PHL when device is removed from operation.

Method	**getDeviceDescription(id : String) : DeviceDescription**
Route	/RTU/{deviceID}
Request method	GET
Description	This method provides the device information stored in MSO. It is used by ORL to know device information to take decisions e.g. to know which services are offered by device for routing the pallet.
Method	**getDevices() : ArrayList<String>**
Route	/RTU
Request method	GET
Description	This method provides a list of URLs for all available devices from which the device information can be extracted or manipulated. The URLs returned are composed using the ID of the devices retrieved from ontology.

4.2.8.2 Order controller module

The order controller module offers services to interact with ORL. OrderController class is also a REST controller in the Spring framework and each method in the class defines a service route. The interaction of OrderController with ORL is shown in the sequence diagram in figure 39. The messages sent to OrderController are the service requests from ORL. The messages between OrderController and MSO represent the queries used to manipulate the information in MSO.

The OrderController class depends on OrderManager class, OntologyManager class and OntoHelper class to fulfill the service requests. The OntologyManager uses the query templates defined in SparqlQueryTemplate class to manipulate the information in MSO to fulfill the request. Order, Pallet and Recipe classes are mapping classes to the JSON message which describes the order, pallet and recipe respectively. The relation between these classes is shown in the figure 40. Similarly OntoHelper also includes query templates which upon retrieving the information can be mapped to Order, Pallet or Recipe classes.

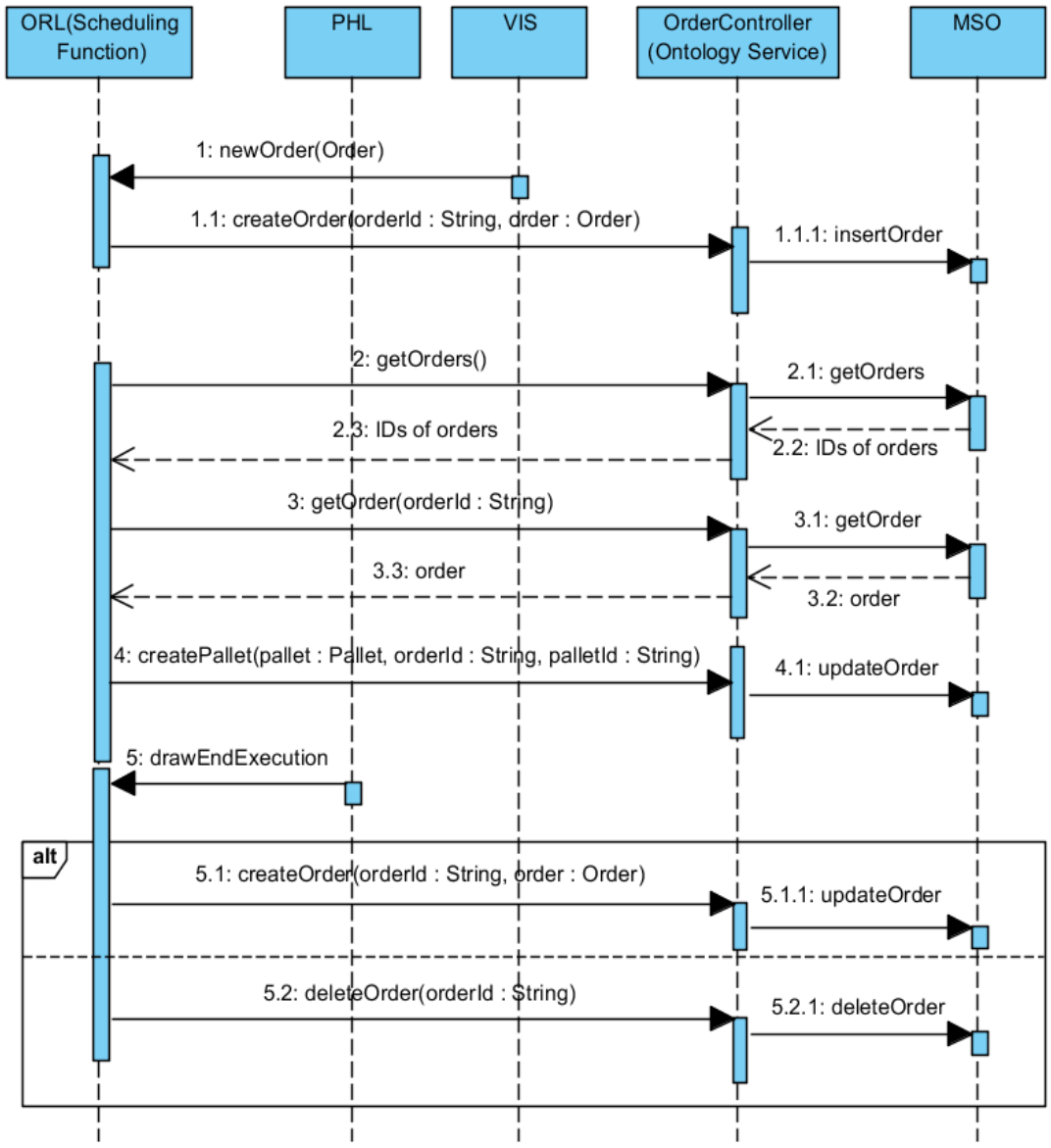


Figure 39. Sequence diagram for OrderController interactions

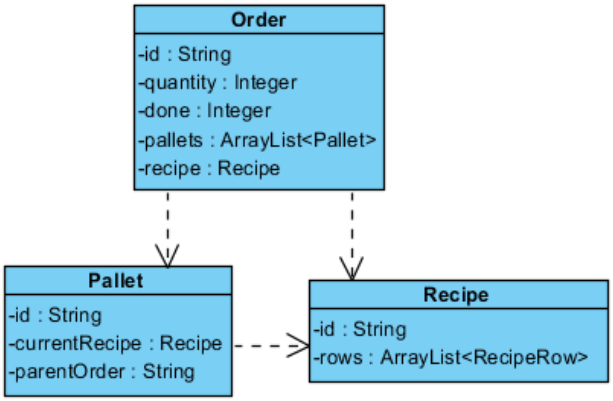


Figure 40. *Class diagram of Order, Recipe and Pallet*

The class diagram of order controller module is shown in figure 41. The description of the methods in OrderController class is shown in table 4. All these services are used by ORL for supporting scheduling and dispatching scenarios shown in figure 29 and 31. The description of the methods in OrderController class is provided in table 5.

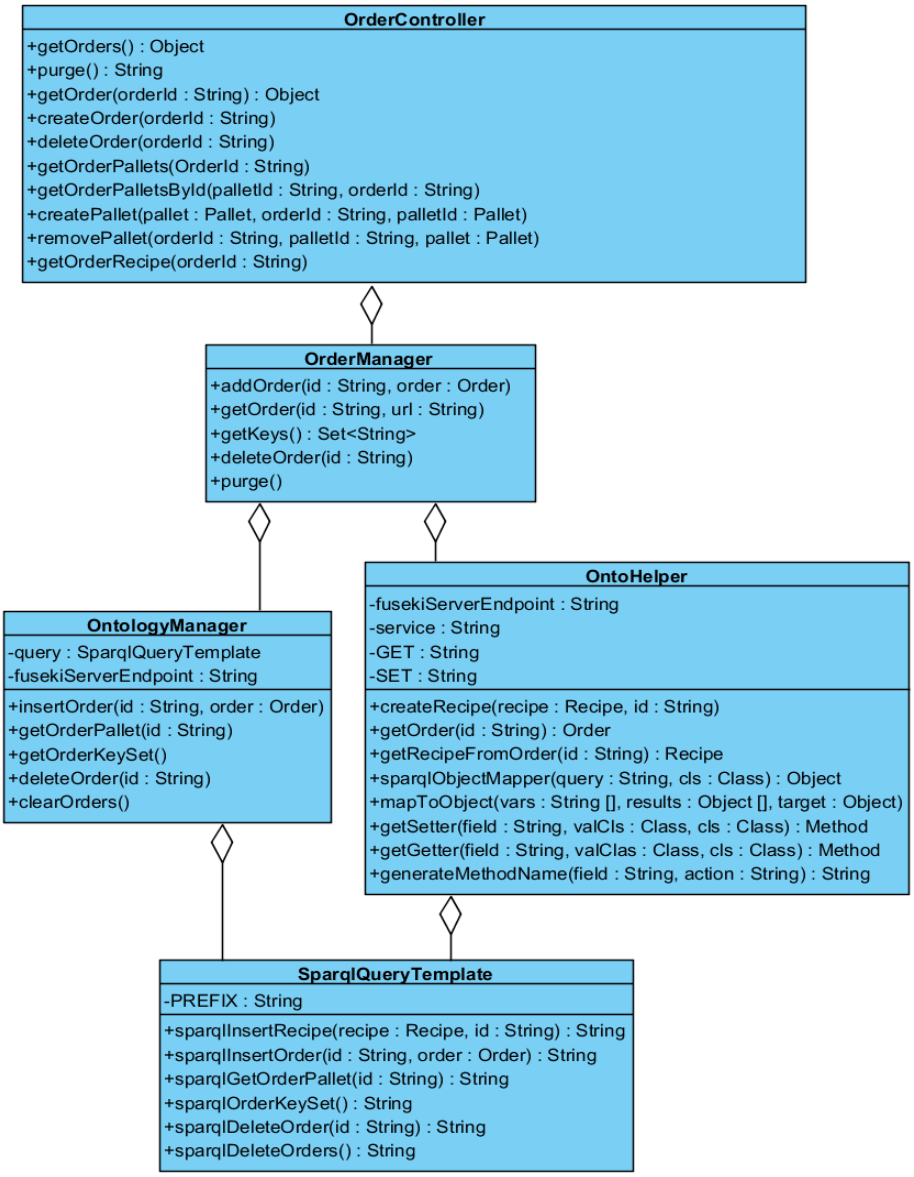


Figure 41. *Class diagram of OrderController module*

Table 5. Description of methods in OrderController class

Method	**getOrders()**
Route	/order
Request method	GET

Description	This method is used to get all available orders from MSO. The service is used by ORL to know what orders are available in order to assign pallets to orders.
Method	**purge()**
Route	/order
Request method	DELETE
Description	This method is used to delete all orders from MSO.
Method	**getOrder(orderId : String) : Object**
Route	/order/{orderId}
Request method	GET
Description	This method is used to get order information from MSO based on order ID.
Method	**createOrder(orderId : String)**
Route	/order/{orderId}
Request method	POST
Description	This method is used to post new order information to MSO
Method	**deleteOrder(orderId : String)**
Route	/order/{orderId}
Request method	DELETE
Description	This method is used to delete order from MSO
Method	**createPallet(pallet : Pallet, palletId : String, orderId : String)**
Route	/order/{orderId}/pallet/{palletId}
Request method	POST
Description	This method is used to assign pallet to an order. This method uses PalletManager class to insert new pallet in MSO and updates order with the new pallet using sparqlInsertOrder method.

Method	**removePallet(pallet : Pallet, palletId : String, orderId : String)**
Route	/order/{orderId}/pallet/{palletId}
Request method	DELETE
Description	This method is used to delete pallet assigned to an order. This method uses PalletManager class to delete the pallet information from MSO and updates order by removing the pallet information using sparqlInsertOrder class.
Method	**getOrderRecipe(orderId : String) : Object**
Route	/order/recipe
Request method	GET
Description	This method is used to get the recipe for the order

4.2.8.3 Pallet controller module

The pallet controller module also offers services for ORL. PalletController class is also a REST controller with each of its methods defining a service route. The interaction of PalletController with ORL is shown in the sequence diagram in figure 42. The messages sent to PalletController are the service requests from ORL. The messages between PalletController and MSO represent the queries used to manipulate the information in MSO.

The PalletController class depends on OntologyManager class and OntoHelper class as like OrderController class to fulfill the service requests. The class diagram of pallet controller module is shown in figure 43. The description of the methods in PalletController class is shown in table 6. The services are used by ORL for supporting scheduling and dispatching scenarios.

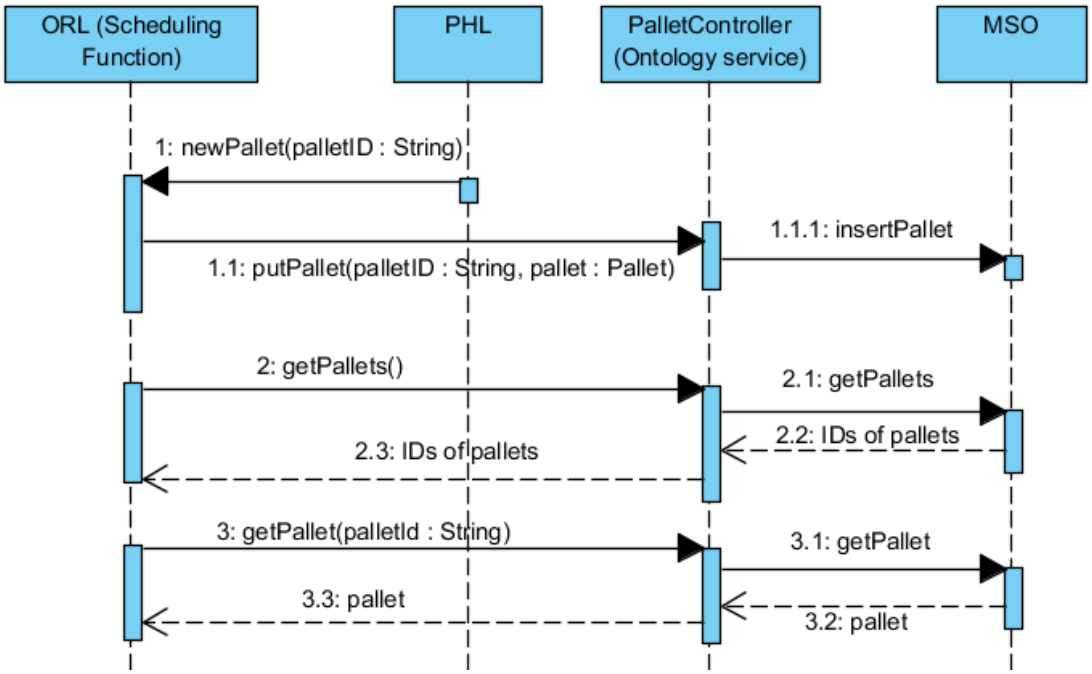


Figure 42. Sequence diagram for PalletController interactions

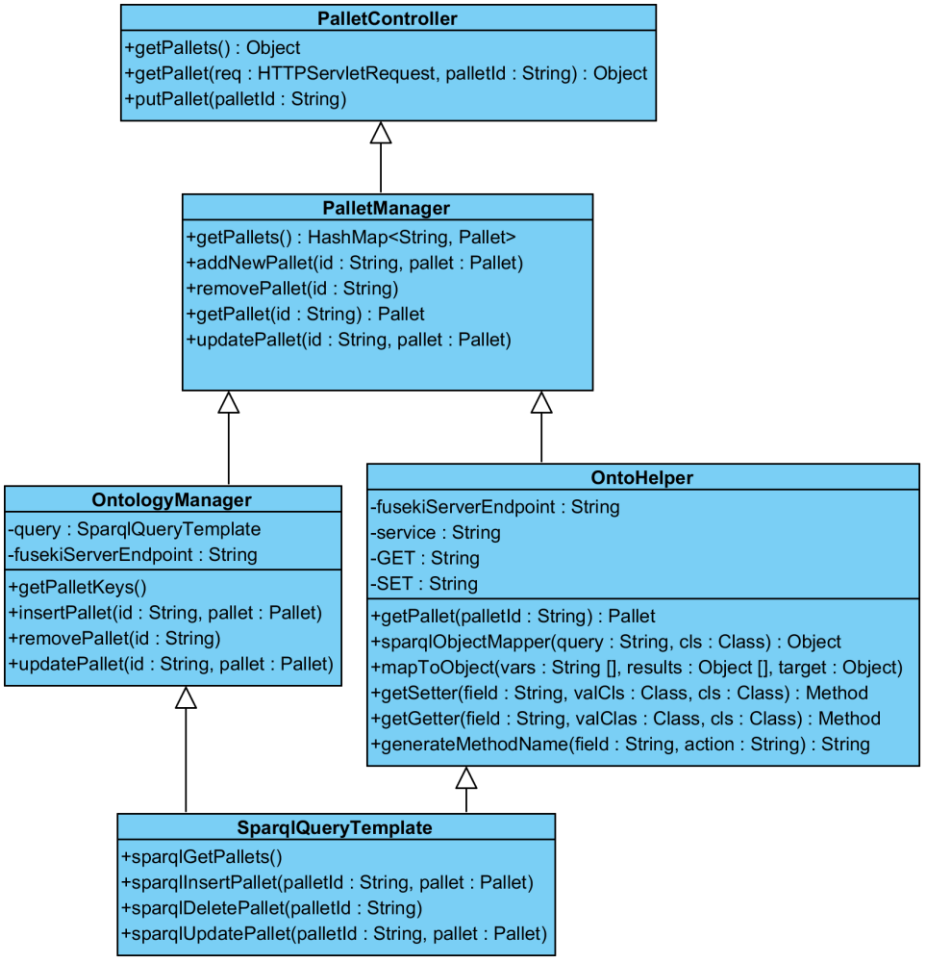


Figure 43. *Class diagram of PalletController module*

Table 6. Description of methods in PalletController class

Method	**getPallets() : Object**
Route	/pallet
Request method	GET
Description	This method is used to get all available pallets from MSO.
Method	**getPallet(palletId : String) : Object**
Route	/pallet/{palletId}
Request method	GET
Description	This method is used to get the pallet information based on the pallet ID.
Method	**putPallet(palletId : String)**
Route	/pallet/{palletId : String}
Request method	POST
Description	This method is used to insert the pallet information to MSO.

4.2.8.4 Visualization provider module

Visualization provider module offers services to interact with the visualization layer (VIS). The interaction of VisualizationProvider with VIS was shown in the sequence diagram in figure 33. The messages sent to VisualizationProvider are the service requests from VIS. The messages between VisualizationProvider and MSO represent the queries used to manipulate the information in MSO.

The services include providing the graphJSON message for creating visualization configuration screen, updating MSO with metadata obtained after screen configuration and providing visualization information of graphical elements (e.g. screen, conveyors, etc.,). The class diagram of this module is illustrated in Figure 41. VisualizationProvider class is a RestController and the service routes are represented by the methods of this class. The VisualizationProvider class uses the methods of VisualizationService class. It works with OntologyManager class to serve the request. SparqlQueryTemplate class holds all the query templates required to manipulate the ontology. The description of the methods in the VisualizationProvider class is shown in table 7.

Table 7. Description of methods in VisualizationProvider class

Method	**getGraphJSON(Id : String) : String**
Route	/vis/configure/{id}
Request method	GET
Description	This method is used to get the visualization screen information in MSO and provides it in GraphJSON format as discussed in section 4.5. This service is used by VIS for creating configuration screen.
Method	**setGraphJSON(Id : String, graphJSON : String)**
Route	/vis/configure/{id}
Request method	POST
Description	This method receives the updated graphJSON message with metadata information from VIS after initial configuration of screen. The method updates the metadata in MSO.
Method	**getElement(Id : String) : String**
Route	/vis/{id}
Request method	GET
Description	This method is used to get the visualization information of graphical element from MSO based on their ID.

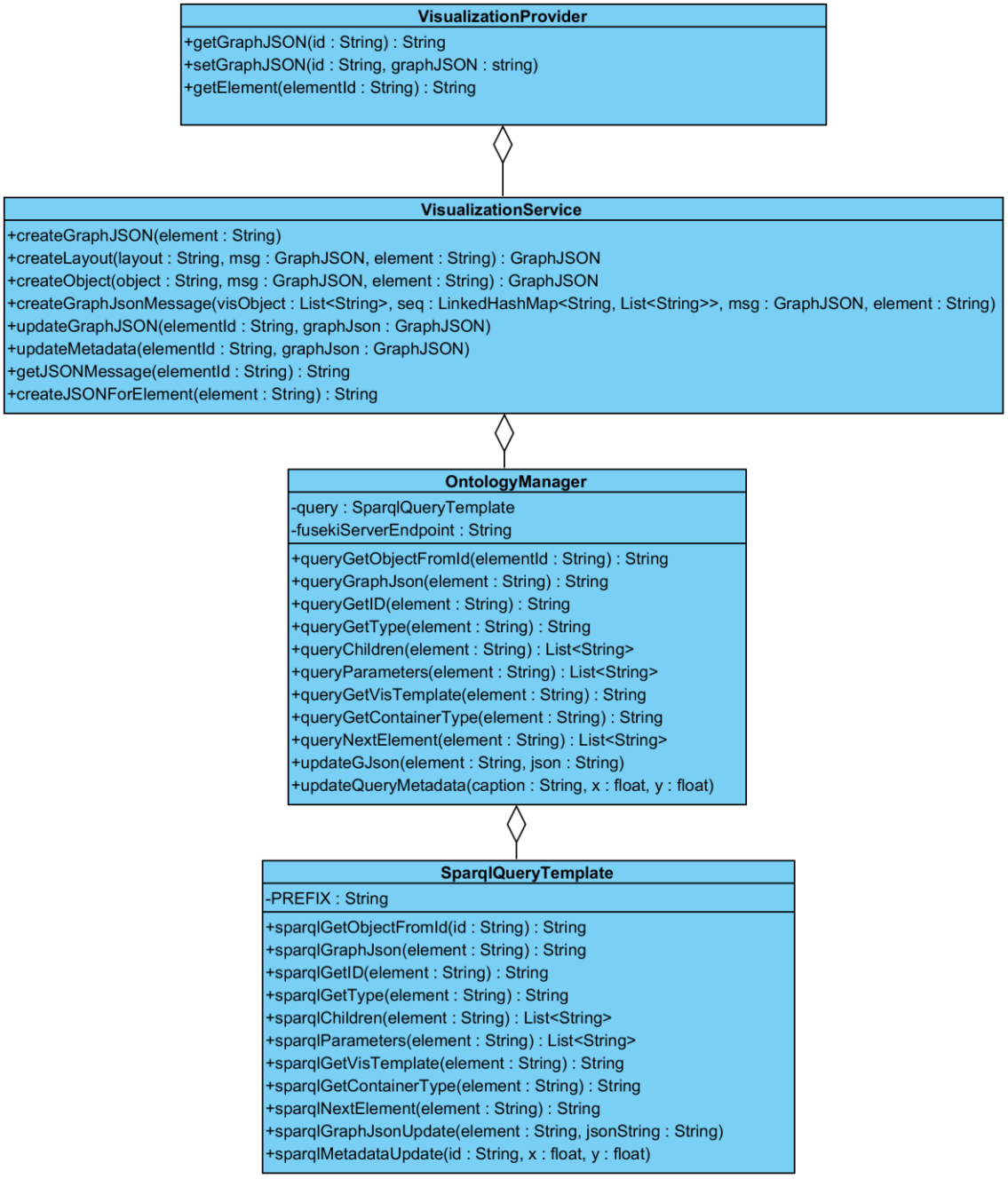


Figure 44. *Class diagram of VisualizationProvider module*

4.3 Summary

At first the ontology is developed following the step 1 to step 5 of the ontology design methodology. After the ontology is developed it is tested for its capability on a real manufacturing system. The different layers of the system architecture are analyzed and the interactions between them are modeled using UML sequence diagrams. The interactions between ontology services and MSO in the sequence diagrams represent the queries that are needed to manipulate the MSO. Based on the interactions the query templates are developed. With this the ontology design process is completed.

Finally, the ontology services are created to insert/ update the information in ontology and expose the knowledge stored in ontology to other layers. The ontology services are implemented using internal modules in order to group the services offered to each layer for easier handling of information in MSO. Each ontology service is mapped to the corresponding query templates that are needed to fulfill the service request. Thus the services are offered for PHL, ORL and VIS in order to support device configuration management, orchestration of processes and re-configurable visualization respectively.

5. DISCUSSION

The main aim of the thesis is to address and solve the key challenges involved in the configuration management, run-time decision making and re-configurable visualization (for monitoring purpose) for manufacturing systems. It makes the manufacturing system to respond readily to rapidly changing customer and market requirements. An ontological approach is used to achieve the objectives of this thesis work. A knowledge model is used to store all the information about the manufacturing system that is necessary to support dynamic re-configuration, run-time decision making and visualization of the system.

The ontological approach is developed in the thesis for a manufacturing system with service-oriented architecture. In order to make available the knowledge about different hierarchical levels of factory, service-oriented architecture is chosen. In this thesis, SOA is deployed using RESTful Web Services. Each layer of the architecture exposes the information in the form of web service. The RPL acts as an integrator layer which provides the ontology service to store the information about all layers in ontology and provide the layers with required knowledge for the execution of the system.

The existing ontologies for SOA based systems are mostly either upper ontologies or focused on specific areas of manufacturing domains. The thesis offers an integrated approach for building ontology with broader capabilities. It combines the concepts in existing ontologies and models the concepts from the perspective of different layers in system architecture. The UML sequence diagram is chosen to model the interaction between different layers. It shows the flow of logic within the system. The interaction between different layers and RPL is used to define the ontology services and query templates. SPARQL queries are chosen to provide the required reasoning for ontologies. With SPARQL, query templates can be created to retrieve, update, create and delete the RDF data in ontology. The ontology services use query templates to fulfill the service request from other layers. For the implementation of the thesis, the ontology services have been developed as RESTful web services in a JAVA programming environment. To extract the information from ontology to Java objects and vice versa Apache JENA is used.

Configuration management is achieved through synchronization of the information system with factory shop-floor. The following solution is offered in this thesis to achieve dynamic re-configuration. The information about the factory shop-floor devices is stored in ontology and updated at run-time based on changes in configuration. For this, shop-floor devices offer self-descriptions (i.e. device information) using RESTful web

services upon their initialization. The description of the device is thus stored in ontology. Likewise when a new device is added to system at run-time, it also offers self-descriptions upon initialization. The re-usability offered by ontology helps to define the new device using existing device representations. Thus new device information is added to ontology at run-time. Likewise the device can delete its information from ontology using web service before being removed from the system. Thus configuration management is achieved through synchronization of factory shop-floor with ontology using the discussed approach.

Run-time decision taking capability can be achieved by orchestration engines by having the control parameters readily available. The control parameters include processes offered by equipment, current location of product etc. which are essential for the orchestration process. This information is also exposed using web services from physical layer and represented in ontology. The enterprise information such as orders, recipe etc. are also essential for orchestration engine to execute the business logic involved in scheduling and dispatching. The approach put forward in this thesis models the enterprise information to support orchestration. The orchestration engine implements the business logic by querying the necessary knowledge from ontology and invoking a sequence of web service operations to control the shop-floor devices. Thus the run-time decision taking capability is achieved by representing the knowledge of control parameters and enterprise information in ontology and providing the information to orchestration engines using web service.

The existing visualization solutions using web technologies have limitation in offering completely re-configurable UIs. Only few details like monitoring parameters, animation of movement of pallets, etc., can be updated dynamically and most of the screen elements are fixed. The approach proposed in this thesis work uses modern web technologies together with ontology as a knowledge base for visualization information to generate UIs. The thesis work proposes the following solution to make visualization reconfigurable. The elements that compose the visualization screen are defined in ontology. In case of visualization of shop-floor layout, ontology provides the required mapping between the actual shop-floor devices and screen elements that represent them. As discussed earlier, the shop-floor devices offer self-descriptions using RESTful web service. These descriptions are stored in ontology and mapped with the screen elements. When new devices are added/ removed from the system, the information in ontology is updated and thereby reflecting the changes in visualization. Thus solution offered in the thesis provides dynamic visualization which is completely integrated to system, reconfigurable and easily maintainable

6. CONCLUSIONS

In this chapter, the conclusions of the thesis and possible future work to refine the proposed solution are presented.

6.1 Thesis conclusions

The contemporary manufacturing systems are expected to respond readily to changing customer and market requirements owing to the highly competitive environment. Hence ability for dynamic re-configuration based on changing requirements and run-time decision making capability are crucial for contemporary manufacturing systems. In order to achieve these objectives a software level re-configuration is needed to keep the information system synchronized with changes in manufacturing system. Software level re-configuration can be achieved only through high level of expertise i.e. knowledge about the manufacturing system. By exposing the information about the shop-floor devices and using algorithms to process the information, one can achieve the above mentioned objectives. Such an approach using ontology as knowledge base for manufacturing system was developed in this thesis.

Knowledge extraction can be achieved by using SOA. RESTful web services technology brings the functionality of web services to device level and offers capability to expose device descriptions to other levels. The combination of SOA and ontology can facilitate effective management of system information. The ontology design methodology developed in the thesis models the concepts from perspective of different layers in the system architecture to build ontology with broader capabilities. The ontology should represent device information (physical layer), enterprise information needed for orchestration of processes (orchestration layer) and visualization information for monitoring (visualization layer) in order to characterize the perspective of the different layers. The required reasoning for ontology can be provided using SPARQL queries. Query templates can be created by analyzing the interactions between different layers and ontology. To keep the information in knowledge base synchronized with the system, the different layers should be able to modify the ontology at run-time based on the changes in manufacturing environment. Using web service technology, ontology services can be provided for other layers to retrieve or update the ontology by making use of queries. Thus by using the technologies and ontology model mentioned above the configuration management and support for run-time execution of the system is achieved.

The proposed approach offers the advantage of system extendibility. The ontology design process is iterative and can be easily extended in case of physical extensions, extension with regard enterprise information or extension in visualization.

The approach developed in the thesis exploits full potential of run-time configuration management, decision taking and re-configuration of visualization UIs. This improves the system adaptability with respect to changing manufacturing environment such as change in equipment or processes. The approach also offers the advantage of reusability of the ontology model. A generic manufacturing system model is developed using the approach and thus it can be adaptable to different kinds of manufacturing systems.

6.2 Future work

The ontological approach put forward in the thesis was implemented on a real manufacturing system and the objectives of the thesis were achieved successfully. However, the possible future works to improve the proposed solution are presented below.

The ontology developed in this thesis has modeled the concepts from perspective of each layer in the architecture. Though the developed model is generic with broader capabilities, it can be further refined by considering the adaptation in different kinds of manufacturing systems.

Another possible future work can be to use reasoners. In the proposed solution, SPARQL queries are used to modify the ontology. In future a set of rules can be added and the model can be inferred using reasoners. SWRL offers a set of rules to express the rules and logic. With the help of reasoners the consistency of the ontology model can be checked when new instances are added to ontology.

REFERENCES

[1] "Advancing Manufacturing paves way for future of industry in Europe." [Online]. Available: http://europa.eu/rapid/press-release_MEMO-14-193_en.htm. [Accessed: 20-Apr-2015].

[2] "EU industrial structure report 2013: Competing in Global Value Chains." [Online]. Available: http://ec.europa.eu/enterprise/policies/industrial-competitiveness/competitiveness-analysis/eu-industrial-structure/files/report_euis_2013_final.pdf. [Accessed: 20-Apr-2015].

[3] "Industrial revolution brings industry back to Europe." [Online]. Available: http://europa.eu/rapid/press-release_IP-12-1085_en.htm?locale=en. [Accessed: 20-Apr-2015].

[4] Hvam, Lars, Mortensen, N. Henrik, Riis, Jesper, *Product Customization*. Berlin, Heidelberg: Springer, 2008.

[5] O. Sauer, "Information Technology for the Factory of the Future – State of the Art and Need for Action," *Procedia CIRP*, vol. 25, pp. 293–296, 2014.

[6] G. Moritz, E. Zeeb, S. Prüter, F. Golatowski, D. Timmermann, and R. Stoll, "Devices Profile for Web Services and the REST," in *2010 8th IEEE International Conference on Industrial Informatics (INDIN)*, 2010, pp. 584–591.

[7] K. Ake, J. Clemons, M. Cubine, and B. Lilly, *Information Technology for Manufacturing: Reducing Costs and Expanding Capabilities*. CRC Press, 2003

[8] "eScop | Artemis project about Embedded systems Service-based Control for Open manufacturing and Process automation." [Online]. Available: http://www.escop-project.eu/. [Accessed: 23-Aug-2015].

[9] B. Vogel-Heuser, G. Kegel, K. Bender, K. Wucherer, Global information architecture for industrial automation. atp 1-2.2009, pp. 108-115.

[10] A. Lobov, J. Puttonen, V. V. Herrera, R. Andiappan, and J. L. M. Lastra, "Service oriented architecture in developing of loosely-coupled manufacturing systems," in *6th IEEE International Conference on Industrial Informatics, 2008. INDIN 2008*, 2008, pp. 791–796.

[11] A. V. Ramos, I. M. Delamer, and J. L. M. Lastra, "Embedded service oriented monitoring, diagnostics and control: Towards the asset-aware and self-recovery factory," in *2011 9th IEEE International Conference on Industrial Informatics (INDIN)*, 2011, pp. 497–502.

[12] R. J. Brachman, *Knowledge Representation and Reasoning*, 1 edition. Amsterdam ; Boston: Morgan Kaufmann, 2004.

[13] Y. Zhang, X. Luo, H. Zhang, and J. W. Sutherland, "A knowledge representation for unit manufacturing processes," *The International Journal of Advanced Manufacturing Technology*, vol. 73, no. 5–8, pp. 1011–1031, May 2014.

[14] M. K. Uddin, A. Dvoryanchikova, A. Lobov, and J. L. M. Lastra, "An ontology-based semantic foundation for flexible manufacturing systems," in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, 2011, pp. 340–345.

[15] M. Stenmark, J. Malec, K. Nilsson, and A. Robertsson, "On Distributed Knowledge Bases for Robotized Small-Batch Assembly," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 519–528, Apr. 2015

[16] J. Krumeich, D. Werth, P. Loos, J. Schimmelpfennig, and S. Jacobi, "Advanced planning and control of manufacturing processes in steel industry through big data analytics: Case study and architecture proposal," in *2014 IEEE International Conference on Big Data (Big Data)*, 2014, pp. 16–24.

[17] L. Fumagalli, S. Pala, M. Garetti, and E. Negri, "Ontology-Based Modeling of Manufacturing and Logistics Systems for a New MES Architecture," in *Advances in Production Management Systems. Innovative and Knowledge-Based Production Management in a Global-Local World*, B. Grabot, B. Vallespir, S. Gomes, A. Bouras, and D. Kiritsis, Eds. Springer Berlin Heidelberg, 2014, pp. 192–200.

[18] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?," *International Journal of Human-Computer Studies*, vol. 43, no. 5–6, pp. 907–928, Nov. 1995.

[19] A. Maedche and S. Staab, "Ontology Learning for the Semantic Web," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 72–79, Mar. 2001.

[20] "Theoretical Foundations of Ontologies," in *Ontological Engineering*, Springer London, 2004, pp. 1–45.

[21] J. L. M. Lastra, I. M. Delamer, and F. Ubis, *Domain Ontologies for Reasoning Machines in Factory Automation*. ISA, 2010.

[22] A. Gómez-Pérez, M. Fernández-López, O. Corcho, *Ontological Engineering*. London: Springer-Verlag, 2004.

[23] S. Borgo and P. Leitão, “Foundations for a Core Ontology of Manufacturing,” in *Ontologies*, R. Sharman, R. Kishore, and R. Ramesh, Eds. Springer US, 2007, pp. 751–775.

[24] A. Gómez-Pérez and O. Corcho, “Ontology Specification Languages for the Semantic Web,” *IEEE Intelligent Systems*, vol. 17, no. 1, pp. 54–60, 2002.

[25] G. Antoniou, E. Franconi, and F. van Harmelen, “Introduction to Semantic Web Ontology Languages,” in *Reasoning Web*, N. Eisinger and J. Małuszyński, Eds. Springer Berlin Heidelberg, 2005, pp. 1–21.

[26] “OWL Web Ontology Language Overview.” [Online]. Available: http://www.w3.org/TR/owl-features/. [Accessed: 23-Aug-2015].

[27] M. Fernández-López, A. Gómez-Pérez, and N. Juristo, “METHONTOLOGY: From Ontological Art Towards Ontological Engineering,” in *Proceedings of the Ontological Engineering AAAI-97 Spring Symposium Series*, Stanford University, EEUU, 1997.

[28] A. De Nicola, M. Missikoff, and R. Navigli, “A software engineering approach to ontology building,” *Information Systems*, vol. 34, no. 2, pp. 258–275, Apr. 2009.

[29] J. I. Olszewska, R. Simpson, and T. L. McCluskey, “Dynamic OWL Ontology Design Using UML and BPMN,” presented at the International Conference on Knowledge Engineering and Ontology Development, 2014, pp. 436–444.

[30] M. Fernandez-Lopez, “Overview of Methodologies for Building Ontologies, ” *In Proceedings of the IJCAI*-99 workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden, 2 August, 1999.

[31] H. Beck and H.S Pinto, “Overview of Approach, Methodologies, Standards, and Tools for Ontologies,”Agricultural Ontology Service (UNFAO), 2003.

[32] Comparative Research on Methodologies for Domain Ontology Development

[33] Achieving Maturity: the State of Practice in Ontology Engineering in 2009

[34] Research of Transactions Goods Domain Ontology Building Method Based on Reputation Recommendation Trust Model

[35] Ontology development (OWL&UML) methodology of web- based Decision Support System for water management

[36] “SPARQL Query Language for RDF.” [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/. [Accessed: 23-Aug-2015].

[37] “SPARQL 1.1 Update.” [Online]. Available: http://www.w3.org/TR/sparql11-update/. [Accessed: 23-Aug-2015].

[38] N. Kumar and S. Kumar, “Querying RDF and OWL data source using SPARQL,” in 2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT), 2013, pp. 1–6.

[39] J. L. M. Lastra and I. M. Delamer, “Ontologies for Production Automation,” in *Advances in Web Semantics I*, T. S. Dillon, E. Chang, R. Meersman, and K. Sycara, Eds. Springer Berlin Heidelberg, 2008, pp. 276–289.

[40] W. Sun, Q.-Y. Ma, T.-Y. Gao, H. Wang, and L. Guo, “Applications of Semantic Web Technologies for Ontology-Based Knowledge Management in Product Development,” in *4th International Conference on Wireless Communications, Networking and Mobile Computing, 2008. WiCOM ’08*, 2008, pp. 1–4.

[41] K. R. C. Koswatte, I. Paik, B. T. G. S. Kumara, and T. H. A. S. Siriweera, “Meta-ontology for innovative product design with semantic TRIZ,” in *2015 International Conference on Electrical and Information Technologies (ICEIT)*, 2015, pp. 379–384.

[42] J. Ge, C. Yang, T. Duan, and Y. Chen, “Research on method of construction and configuration of product family based on ontology,” in *2013 International Conference on Measurement, Information and Control (ICMIC)*, 2013, vol. 02, pp. 1436–1440.

[43] J. Chen, Y. Cheng, and D. Nie, “Product configuration method based on ontology mapping,” in *2013 4th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2013, pp. 97–101.

[44] Z. Huang, L. Qiao, N. Anwer, and Y. Mo, “Ontology Model for Assembly Process Planning Knowledge,” in *Proceedings of the 21st International Conference on Industrial Engineering and Engineering Management 2014*, E. Qi, J. Shen, and R. Dou, Eds. Atlantis Press, 2015, pp. 419–423.

[45] S. Bussmann, N. R. Jennings, and M. Wooldridge, *Multiagent Systems for Manufacturing Control: A design Methodology*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[46] Vladimír Marík, Pavel Vrba, Paulo Leitão, *Holonic and Multi-Agent Systems for Manufacturing: 5th International Conference on Industrial Applications of Holonic and Multi-Agent Systems, HoloMAS ...*, 2011 edition. New York: Springer, 2011.

[47] M. R. Pavel Vrba, “Semantic technologies: latest advances in agent-based manufacturing control systems,” *International Journal of Production Research*, vol. 49, pp. 1483–1496, 2011.

[48] “Welcome to the Foundation for Intelligent Physical Agents.” [Online]. Available: http://www.fipa.org/. [Accessed: 23-Aug-2015].

[49] M. Obitko and V. Marik, “Ontologies for multi-agent systems in manufacturing domain,” in *13th International Workshop on Database and Expert Systems Applications, 2002. Proceedings*, 2002, pp. 597–602.

[50] S. Borgo and P. Leitão, “The Role of Foundational Ontologies in Manufacturing Domain Applications,” in *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, R. Meersman and Z. Tari, Eds. Springer Berlin Heidelberg, 2004, pp. 670–688.

[51] L. Ferrarini, C. Veber, A. Luder, J. Peschke, A. Kalogeras, J. Gialelis, J. Rode, D. Wunsch, and V. Chapurlat, “Control Architecture for Reconfigurable Manufacturing Systems: the PABADIS’PROMISE approach,” in *IEEE Conference on Emerging Technologies and Factory Automation, 2006. ETFA ’06*, 2006, pp. 545–552.

[52] C. Alexakos, M. Georgoudakis, A. P. Kalogeras, and S. Likothanassis, “Adaptive manufacturing utilizing ontology-driven multi-agent systems: Extending Pabadis’ Promise approach,” in *2012 IEEE International Conference on Industrial Technology (ICIT)*, 2012, pp. 42–47.

[53] U. Kannengiesser and H. Muller, “Towards Agent-Based Smart Factories: A Subject-Oriented Modeling Approach,” in *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, 2013, vol. 3, pp. 83–86.

[54] M. Sorouri, V. Vyatkin, and Z. Salcic, “MIRA: Enabler of mass customization through agent-based development of intelligent manufacturing systems,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 258–263.

[55] M. Merdan, P. Vrba, G. Koppensteiner, and A. Zoitl, “Knowledge-based multi-agent architecture for dynamic scheduling in manufacturing systems,” in *6th IEEE International Conference on Industrial Informatics, 2008. INDIN 2008*, 2008, pp. 1075–1080.

[56] A. Phutthisathian, N. Maneerat, R. Varakulsiripunth, K. Takahashi, and Y. Kato, “An ontology-based multi agent autotnotive parts transportation management sys-

tem," in *Humanitarian Technology Conference (R10-HTC), 2013 IEEE Region 10*, 2013, pp. 96–99.

[57] K. Nagorny, J. Barata, and A. W. Colombo, "A survey of service-based systems-of-systems manufacturing systems related to product life-cycle support and energy efficiency," in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, 2014, pp. 582–587.

[58] B. Ramis, L. Gonzalez, S. Iarovyi, A. Lobov, J. L. Martinez Lastra, V. Vyatkin, and W. Dai, "Knowledge-based web service integration for industrial automation," in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, 2014, pp. 733–739.

[59] I. M. Delamer and J. L. M. Lastra, "Ontology Modeling of Assembly Processes and Systems using Semantic Web Services," in *2006 IEEE International Conference on Industrial Informatics*, 2006, pp. 611–617.

[60] J. Puttonen, A. Lobov, and J. L. Martinez Lastra, "Semantics-Based Composition of Factory Automation Processes Encapsulated by Web Services," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2349–2359, Nov. 2013.

[61] "Socrades Project 2006-2009." [Online]. Available: http://www.socrades.net/. [Accessed: 23-Aug-2015].

[62] M. L. Sbodio, D. Martin, and C. Moulin, "Discovering Semantic Web services using SPARQL and intelligent agents," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 8, no. 4, pp. 310–328, Nov. 2010.

[63] P. Vrba, P. Kadera, V. Jirkovský, M. Obitko, and V. Mařík, "New Trends of Visualization in Smart Production Control Systems," in *Holonic and Multi-Agent Systems for Manufacturing*, V. Mařík, P. Vrba, and P. Leitão, Eds. Springer Berlin Heidelberg, 2011, pp. 72–83.

[64] P. Vrba, P. Tichý, V. Mařík, K. H. Hall, R. J. Staron, F. P. Maturana, and P. Kadera, "Rockwell Automation's Holonic and Multiagent Control Systems Compendium," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 41, no. 1, pp. 14–30, Jan. 2011.

[65] I. M. Delamer and J. L. M. Lastra, "Self-orchestration and choreography: towards architecture-agnostic manufacturing systems," in *20th International Conference on Advanced Information Networking and Applications, 2006. AINA 2006*, 2006, vol. 2, p. 5 pp.–.

[66] A. N. Lee and J. L. M. Lastra, “Data aggregation at field device level for industrial ambient monitoring using Web Services,” in *2011 9th IEEE International Conference on Industrial Informatics (INDIN)*, 2011, pp. 491–496.

[67] “puttonen.pdf” [Online]. Available: http://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/22371/puttonen.pdf?sequence=3&isAllowed=y. [Accessed: 14-Sep-2015]

[68] “FASTory Simulator (eScop)” [Online]. Available: http://escop.rd.tut.fi:3000/. [Accessed: 20-Sep-2015].